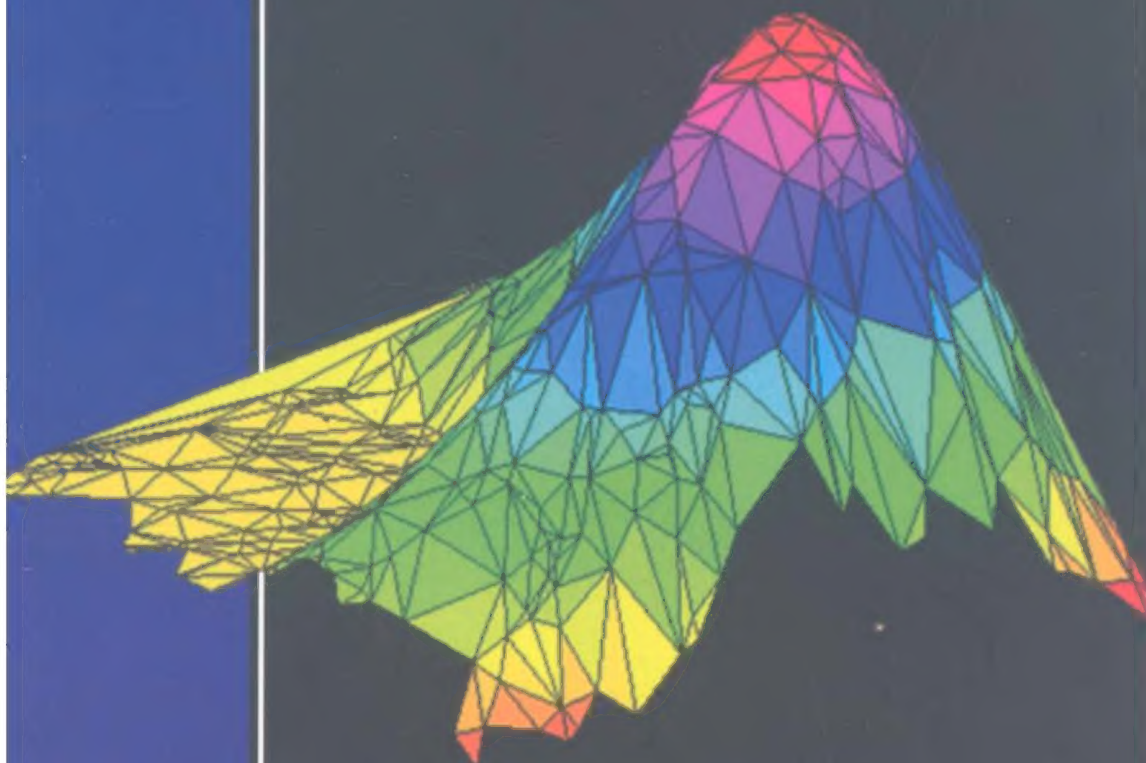


LẬP TRÌNH MATLAB VÀ ỨNG DỤNG

(Dùng cho sinh viên khối khoa học và kỹ thuật)



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
THS. NGUYỄN HOÀNG HẢI - THS. NGUYỄN VIỆT ANH

LẬP TRÌNH MATLAB VÀ ỨNG DỤNG

Dành cho sinh viên khối khoa học và kỹ thuật

In lần thứ 4 có sửa chữa và bổ sung



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

Hà Nội 2006

MỤC LỤC

	Trang
Lời giới thiệu	7
Cài đặt MATLAB cho WINDOWS	9
Chương 1 Giới thiệu chung	11
1.1. Các phép toán đơn giản	12
1.2. Không gian làm việc của MATLAB	14
1.3. Biến	15
1.4. Câu giải thích (comment) và sự chấm câu	17
1.5. Số phức	18
Chương 2 Các bài toán kỹ thuật	21
2.1. Các hàm toán học thông thường	21
2.2. Các ví dụ	24
Chương 3 Những đặc điểm của cửa sổ lệnh trong MATLAB	30
3.1. Quản lý không gian làm việc của MATLAB	30
3.2. Ghi và phục hồi dữ liệu	32
3.3. Khuôn dạng hiển thị số	33
Chương 4 Script M_files	35
Chương 5 Quản lý tệp	40
Chương 6 Các phép toán đối với mảng	45
6.1. Mảng đơn	45
6.2. Địa chỉ của mảng	46
6.3. Cấu trúc của mảng	47
6.4. Vector hàng và vector cột	50
6.5. Mảng có các phần tử là 0 hoặc 1	57
6.6. Thao tác đối với mảng	58
6.7. Tìm kiếm mảng con	64
6.8. So sánh mảng	66
6.9. Kích cỡ của mảng	70
6.10. Mảng nhiều chiều	72
Chương 7 Các thao tác với mảng	78
7.1. Tạo phương trình tuyến tính	78
7.2. Các hàm ma trận	82
7.3. Ma trận đặc biệt	83
Chương 8 Các phép tính logic và quan hệ	89
8.1. Toán tử quan hệ	89
8.2. Toán tử logic	91
8.3. Các hàm logic và hàm quan hệ	92
Chương 9 Văn bản	94
9.1. Xâu kí tự	94
9.2. Chuyển đổi xâu	97
9.3. Các hàm về xâu	98
9.4. Ma trận tế bào của xâu	100
Chương 10 Thời gian	104
10.1. Ngày và giờ hiện tại	104
10.2. Sự chuyển đổi giữa các kiểu	105

10.3. Các hàm về ngày	107
10.4. Các hàm về thời gian	108
10.5. Vẽ đồ thị với hàm ngày và hàm thời gian	110
Chương 11 Vòng lặp điều khiển	114
11.1. Vòng lặp for	114
11.2. Vòng lặp while	117
11.3. Cấu trúc if-else-end	118
11.4. Cấu trúc switch-case	120
Chương 12 Hàm M_FILE	127
12.1. Các quy luật và thuộc tính	128
12.2. Các ví dụ	131
Chương 13 Phân tích dữ liệu	138
Chương 14 Các phép tính đối với đa thức	149
14.1. Các nghiệm của đa thức	149
14.2. Nhân đa thức	150
14.3. Phép cộng đa thức	150
14.4. Chia hai đa thức	152
14.5. Đạo hàm	152
14.6. Tính giá trị của một đa thức	152
14.7. Phân thức hữu tỷ	153
Chương 15 Phép nội suy và mịn hoá đường cong	156
15.1. Mịn hoá đường cong	156
15.2. Nội điểm một chiều	160
15.3. Xấp xỉ hoá hai chiều	164
Chương 16 Phân tích số liệu	168
16.1. Vẽ đồ thị	168
16.2. Cực trị của một hàm	170
16.3. Tìm giá trị không	172
16.4. Phép lấy tích phân	173
16.5. Phép lấy vi phân	174
16.6. Phương trình vi phân	177
Chương 17 Đồ hoạ trong hệ toạ độ phẳng	181
17.1. Sử dụng lệnh Plot	181
17.2. Kiểu đường, dấu và màu	183
17.3. Kiểu đồ thị	184
17.4. Đồ thị lưới, hộp chứa trục, nhãn và lời chú giải	185
17.5. Kiến tạo hệ trục toạ độ	187
17.6. In hình	191
17.7. Thao tác với đồ thị	191
17.8. Một số đặc điểm khác của đồ thị trong hệ toạ độ phẳng	194
Chương 18 Đồ hoạ trong không gian ba chiều	199
18.1. Đồ thị đường thẳng	199
18.2. Đồ thị bề mặt và lưới	201
18.3. Thao tác với đồ thị	203
18.4. Các đặc điểm khác của đồ thị trong không gian ba chiều	206
18.5. Bảng màu	208
18.6. Sử dụng bảng màu	209

18.7. Sử dụng màu để thêm thông tin.	210
18.8. Hiển thị bảng màu	211
18.9. Thiết lập và thay đổi bảng màu	213
Chương 19 Mạng tế bào và cấu trúc	216
19.1. Mạng tế bào	216
19.2. Xây dựng và hiển thị mạng tế bào	216
19.3. Tổ hợp và khôi phục mạng tế bào	217
19.4. Truy nhập vào trong mạng tế bào	218
19.5. Mạng tế bào của chuỗi ký tự	219
19.6. Cấu trúc	220
19.7. Xây dựng mạng cấu trúc	220
19.8. Truy nhập vào các trường cấu trúc	221
19.9. Sự nghịch đảo và hàm kiểm tra	224
Chương 20 Biểu tượng của hộp công cụ toán học	225
20.1. Biểu thức và các đối tượng đặc trưng	225
20.2. Tạo và sử dụng các đối tượng đặc trưng	226
20.3. Sự biểu diễn biểu thức đặc trưng của MATLAB	227
20.4. Biến đặc trưng	232
20.5. Phép toán trên biểu thức đặc trưng	232
20.6. Tách các tử số và mẫu số	232
20.7. Phép toán đại số tiêu chuẩn	234
20.8. Các phép toán nâng cao	236
20.9. Hàm nghịch đảo	237
20.10. Sự thay thế biến số	238
20.11. Phép lấy vi phân	239
20.12. Phép tích phân	240
20.13. Vẽ đồ thị biểu thức đặc trưng	244
20.14. Định dạng và đơn giản hoá	244
20.15. Tóm tắt và một số đặc điểm khác	246
20.16. Tự làm	247
20.17. Giải phương trình	248
20.18. Giải phương trình đại số đơn giản	248
20.19. Một vài phép toán đại số	249
20.20. Phép toán tích phân	250
20.21. Một vài phép toán tích phân	250
20.22. Ma trận và đại số tuyến tính	251
20.23. Phép toán đại số tuyến tính	252
20.24. Hàm bước và xung	253
20.25. Biến đổi Laplace	253
20.26. Biến đổi Fourier	253
Chương 21 Hộp công cụ hệ thống điều khiển	255
21.1. Sự biểu diễn bằng đồ thị	255
21.2. Đối tượng LTI	256
21.3. Khôi phục dữ liệu	259
21.4. Sự nghịch đảo đối tượng	260
21.5. Thuật toán đối tượng LTI	261
21.6. Phân tích hệ thống	261

21.7. Danh sách các hàm của hộp công cụ hệ thống điều khiển	264
Chương 22 Hộp dụng cụ xử lý tín hiệu	269
22.1 Chức năng của hộp công cụ xử lý tín hiệu	269
22.2 Biểu diễn tín hiệu	269
22.3 Các tín hiệu đa kênh	270
22.4 Tạo dạng tín hiệu	271
22.5 Các tín hiệu cơ bản	272
22.6 Các tín hiệu tuần hoàn thông dụng	272
22.7 Các tín hiệu không chu kỳ	274
22.8 Hàm chuỗi xung	274
22.9 Hàm Sinc	275
22.10 Nhập xuất dữ liệu với hộp công cụ	276
22.11 Thực hiện tích chập	276
22.12 Thực hiện lọc trong miền thời gian	277
22.13 Thực hiện bỏ lọc trong miền z	277
22.14 Thực hiện lọc bằng hàm filter	278
22.15 Đáp ứng xung	279
22.16 Đáp ứng tần số	280
22.17 Hiện thị đáp ứng của hệ thống bằng lệnh fvtool	282
22.18 Trễ nhóm và trễ pha	282
22.19 Phân tích điểm cực – không	283
22.20 Phân tích Fourier rời rạc	284
22.21 Mô hình hệ thống tuyến tính rời rạc	285
22.22 Danh sách các hàm của hộp công cụ xử lý tín hiệu	288
Chương 23 Hộp công cụ truyền thống	296
23.1 Nhiễu Gauss trắng	296
23.2 Các ma trận symbol ngẫu nhiên	296
23.3 Tạo các ma trận số nguyên ngẫu nhiên	297
23.4 Các sơ đồ lỗi bit ngẫu nhiên	297
23.5 Tỷ số lỗi	298
23.6 So sánh tỷ số lỗi symbol và tỷ số lỗi bit	299
23.7 Cộng nhiễu trắng Gaussian vào tín hiệu	300
23.8 Các bộ sinh chuỗi giả nhiễu, mặt nạ và giá trị dịch	301
23.9 Mã hóa/giải mã BCH	302
23.10 Bộ giải mã khối	303
23.11 Bộ giải mã khối	305
23.12 Tính khoảng cách cực tiểu của 1 mã khối tuyến tính	308
23.13 Tạo ra ma trận sinh và ma trận kiểm tra chẵn lẻ cho mã Hamming	309
23.14 Sinh ra bảng giải mã syndrome	310
23.15 Đa thức sinh của mã Reed-Solomon	310
23.16 Giải mã chấp dữ liệu nhí phân sử dụng thuật toán Viterbi	312
23.17 Điều chế băng thông tương tự	314
23.18 Giải điều chế băng thông tương tự	317
23.19 Bộ điều chế băng thông số	318
23.20 Bộ điều chế băng gốc số	320
23.21 Bộ giải điều chế băng thông số	322
23.22 Bộ giải điều chế băng gốc số	324
23.23 Phân loại các hàm	329
Chương 24 Trợ giúp	335
24.1. Cửa sổ lệnh trợ giúp	335
24.2. Cửa sổ trợ giúp	338
24.3. Các M_File của Student Edition	340
Tài liệu tham khảo	371

LỜI GIỚI THIỆU

Các nhà khoa học, các kỹ sư, kỹ thuật viên và sinh viên các trường Đại học kỹ thuật luôn quan tâm đến việc phát triển nâng cao khả năng tính toán và xử lý trên máy tính những vấn đề chuyên môn đa dạng trong nghiên cứu khoa học. Dĩ nhiên không phải ai trong số họ cũng là những lập trình viên sử dụng thành thạo các ngôn ngữ lập trình để giải quyết những vấn đề đó trên máy tính.

Matlab (Maxtrix Laboratory) là một công cụ phần mềm của MathWork với giao diện cực mạnh cùng những lợi thế trong kỹ thuật lập trình đáp ứng được những vấn đề hết sức đa dạng: từ các lĩnh vực kỹ thuật chuyên ngành như điện, điện tử, điều khiển tự động, robot công nghiệp, vật lý hạt nhân cho đến các ngành xử lý toán chuyên dụng như thống kê, kế toán v.v... đã giải quyết được những vấn đề nói trên một cách đơn giản, trực quan mà không cần đòi hỏi người sử dụng phải là những lập trình viên chuyên nghiệp.

Matlab cùng bộ lệnh rất mạnh của nó cho phép giải quyết các loại bài toán khác nhau, đặc biệt là các hệ phương trình tuyến tính, phi tuyến hay các bài toán ma trận với kết quả nhanh chóng và chính xác. Bộ lệnh này lên tới hàng trăm và ngày càng được mở rộng thông qua các hàm ứng dụng được tạo lập bởi người sử dụng hay thông qua thư viện trợ giúp. Bên cạnh đó, Matlab cho phép xử lý dữ liệu, biểu diễn đồ họa một cách mềm dẻo, đơn giản và chính xác trong không gian hai chiều cũng như ba chiều giúp người sử dụng có thể quan sát kết quả một cách trực quan và đưa ra giải pháp tốt nhất. Được tích hợp cùng với một số ngôn ngữ lập trình thông dụng khác như C, C++, Fortran, Java v.v... do đó những ứng dụng của Matlab có thể được chuyển đổi một cách dễ dàng, mềm dẻo sang những ngôn ngữ đó. Với hàng loạt những ưu điểm nói trên, Matlab đã, đang và sẽ được sử dụng rộng rãi trên nhiều lĩnh vực cũng như nhiều nước trên toàn thế giới.

Để cung cấp cho bạn đọc một công cụ trợ giúp hữu ích của tin học ứng dụng, chúng tôi giới thiệu cuốn sách "**Lập trình Matlab - Dành cho sinh viên khối khoa học và kỹ thuật**" do nhóm tác giả của bộ môn Hệ thống Viễn thông thuộc Khoa Điện tử Viễn thông biên soạn. Tiêu đề của cuốn sách đã cho thấy rõ đối tượng mà các tác giả hướng tới là những sinh viên đang theo học các trường khoa học kỹ thuật. Tuy nhiên cuốn sách này cũng rất hữu ích cho cả những kỹ sư, cán bộ kỹ thuật hay những nhà khoa học trong việc tra cứu.

Trong lần xuất bản đầu tiên này, mặc dù cũng đã có rất nhiều cố gắng, nhưng cuốn sách cũng còn những vấn đề chưa thể đề cập hết hoặc còn thiếu sót ở một mức độ nào đó. Chúng tôi

rất mong nhận được sự góp ý xây dựng và phê bình chân thành của bạn đọc. Mọi thắc mắc, có thể gửi thư góp ý về địa chỉ sau:

Nguyễn Hoàng Hải - Bộ môn Hệ thống Viễn thông

Khoa Điện tử Viễn thông

Trường Đại học Bách Khoa Hà nội

Email: nhhaijp@mail.hut.edu.vn

Xin chân thành cảm ơn ban biên tập cùng với các bạn đồng nghiệp đã đóng góp và chỉnh sửa để cuốn sách này thêm hoàn thiện hơn và bớt được những lỗi đáng tiếc.

Cuối cùng chúc các bạn trẻ đặc biệt là các bạn sinh viên có được một chuyến du hành thú vị vào thế giới MatLab với nhiều thành công nhất và đóng góp những kết quả nghiên cứu của mình với sự nghiệp công nghiệp hoá, hiện đại hoá đất nước.

Trưởng khoa Điện tử Viễn thông

TS. Phạm Minh Việt

CÀI ĐẶT MATLAB CHO WINDOWS

Yêu cầu hệ thống

- Hệ thống IBM hoặc tương thích 100% với bộ vi xử lý 486 Intel cộng với bộ đồng xử lý toán học 487 (ngoại trừ 486 DX có bộ xử lý bên trong), Pentium hoặc Pentium Pro Processor.
- Microsoft Windows 95 hoặc Window NT.

a) CD ROM

- Bộ điều phối đồ họa 8 bit và card màn hình (256 màu đồng thời)

- Khoảng trống đĩa đủ để cài đặt và chạy các tùy chọn. Sự yêu cầu đĩa cứng thay đổi tùy theo kích cỡ các partition và các tệp trợ giúp được cài đặt trực tiếp theo tùy chọn. Quá trình cài đặt sẽ thông báo cho bạn biết tỉ mỉ về dung lượng đĩa yêu cầu. Ví dụ:

Partition với một liên cung mặt 0 cần 25 MB cho riêng MATLAB và 50 MB cho cả MATLAB và HELP.

Partition với liên cung 64 KB cần 115 MB cho riêng MATLAB và 250 MB cho cả MATLAB và HELP.

b) Bộ nhớ

Microsoft Window 95: 8 MB tối thiểu và 16 MB khuyến nghị.

Microsoft WIN NT 3.51 hoặc 4.0: 12 MB tối thiểu và 16 MB khuyến nghị.

Các khuyến nghị

- Bộ nhớ phụ vào (Bộ nhớ bổ sung: Additional Memory).
- Vĩ mạch tăng tốc đồ họa hỗ trợ cho Microsoft Window.
- Máy in trợ giúp cho Microsoft Window.
- Vĩ mạch âm thanh trợ giúp cho Microsoft Window.
- Microsoft Word 7.0 hoặc hơn (nếu bạn có ý định sử dụng MATLAB NoteBook).
- Trình biên dịch Watcom C, Borland, Microsoft (xây dựng file MEX).

- Netscape Navigator 2.0 hoặc version cao hơn hoặc Microsoft Internet Explorer 3.0 để chạy MATLAB Help Desk.

Quá trình cài đặt

1. Đặt đĩa vào ổ CD. Trên WIN 95 chương trình SETUP bắt đầu chạy tự động nếu như MATLAB chưa được cài từ trước. Còn không, nhấn đúp vào biểu tượng **setup.exe** để bắt đầu quá trình cài đặt.

2. Chấp nhận hay bỏ đi những khuyến cáo về cấp đăng kí phần mềm trên màn hình. Nếu chấp nhận bạn mới có thể bắt đầu quá trình cài đặt.

3. Trên Customer Information, nhập vào tên bạn, địa chỉ của bạn. Tên không được quá 30 kí tự. Nhấn nút NEXT.

4. Nhấn vào các hộp trống thành phần dấu 'v' nếu như bạn muốn tùy chọn đó và nhấn tiếp nếu bạn có ý định không muốn tùy chọn đó (có thể thêm vào sau này nếu muốn). Trên màn hình hiển thị C:\MATLAB là thư mục đích mặc định của quá trình cài đặt. Nếu bạn muốn cài đặt vào thư mục khác hoặc đổi tên thư mục thì bạn lựa chọn Browse.

MATLAB cho Macintosh.

MATLAB cho máy Macintosh chạy được trên:

- Mọi máy Macintosh có cấu hình đủ mạnh (power Macintosh).
- Mọi Macintosh được trang bị bộ vi xử lí 68040 (bộ đồng xử lí toán học bên trong).
- Mọi máy Macintosh được trang bị bộ vi xử lí 68020 hoặc 68030 và bộ đồng xử lí toán học 68881 hoặc 68882.
- Yêu cầu tối thiểu để chạy MATLAB.
- Đĩa cứng trống tối thiểu 26 MB, cần thêm 60 MB cho hệ thống tùy chọn HELP trực tuyến.
- 16 MB cho phân vùng bộ nhớ.
- Ổ CD ROM.
- Color Quick Draw.

GIỚI THIỆU CHUNG

Trong phần này chúng ta sẽ xem xét một số những ứng dụng của MatLab; vì để trình bày tất cả những ứng dụng của MATLAB sẽ rất dài và tốn thời gian. Sau khi đọc quyển hướng dẫn này, bạn sẽ thấy MATLAB là ngôn ngữ rất mạnh để giải quyết những vấn đề quan trọng và khó khăn của bạn. Nó sẽ rất hữu ích khi bạn đọc phần hướng dẫn cơ bản vì nó sẽ cung cấp cho bạn những kiến thức để bạn hiểu rõ MATLAB và phát triển được những khả năng của mình sau này.

Có lẽ cách dễ nhất để hình dung về MATLAB là nó có thể thực hiện các chức năng của máy tính cá nhân: giống như các máy tính cơ bản (Calculator), nó làm tất cả các phép tính toán học cơ bản như cộng, trừ, nhân, chia; giống như máy tính kỹ thuật (Scientific Calculator), nó bao gồm các phép tính: số phức, căn thức, số mũ, logarithm, các phép toán lượng giác như sine, cosine, tang; nó cũng giống như máy tính cá nhân (PC) có khả năng lập trình, có thể lưu trữ, tìm kiếm lại dữ liệu, cũng có thể tạo, bảo vệ và ghi trình tự các lệnh để tự động tính toán khi giải quyết các bài toán, bạn có thể so sánh logic, điều khiển thực hiện lệnh để đảm bảo tính đúng đắn của phép toán. Giống như các máy tính hiện đại nhất, nó cho phép bạn biểu diễn dữ liệu dưới nhiều dạng như: biểu diễn thông thường, ma trận đại số, các hàm tổ hợp và có thể thao tác với dữ liệu thường cũng như đối với ma trận.

Trong thực tế MATLAB còn được ứng dụng rất rộng rãi trong nhiều lĩnh vực và nó cũng được sử dụng rất nhiều để giải các phép tính toán học. Với những đặc điểm đó và khả năng thân thiện với người sử dụng nên nó dễ dàng sử dụng hơn các ngôn ngữ khác như Basic, Pascal, C.

Nó cung cấp một môi trường phong phú cho biểu diễn dữ liệu, và có khả năng mạnh mẽ về đồ họa, bạn có thể tạo các giao diện riêng cho người sử dụng (GUIs) để giải quyết những vấn đề riêng cho mình. Thêm vào đó MATLAB đưa ra những công cụ để giải quyết những vấn đề đặc biệt, gọi là Toolbox (hộp công cụ). Ví dụ Student Edition của MATLAB bao gồm cả Toolbox điều khiển hệ thống, Toolbox xử lý tín hiệu, Toolbox biểu tượng toán học. Ngoài ra bạn có thể tạo Toolbox cho riêng mình.

Với những khả năng mạnh mẽ, rộng lớn của MATLAB và để có thể sử dụng được MatLab bạn nên đầu tư phần cơ bản. Sau đây chúng ta sẽ nghiên cứu từng phần, và cuốn sách này sẽ giúp bạn hiểu được chúng. Trước tiên, một cách đơn giản nhất là chúng ta quan niệm như là một máy tính cơ bản, tiếp theo là như máy tính kỹ thuật và như máy tính có thể lập trình được, cuối cùng là như máy tính hiện đại nhất. Bằng cách quan niệm này bạn sẽ dễ dàng hiểu được những

cách mà MATLAB giải quyết những vấn đề thông thường và xem MATLAB giải quyết những vấn đề về số phức mềm dẻo như thế nào.

Tùy thuộc vào kiến thức của bạn, bạn có thể tìm thấy những phần trong cuốn sách hướng dẫn này hứng thú hay buồn tẻ. Khi bạn chạy chương trình MATLAB, nó sẽ tạo một hoặc nhiều cửa sổ trên màn hình của bạn, và cửa sổ lệnh (command) là cửa sổ chính để bạn giao tiếp với MATLAB.

Các ký tự "EDU>>" là dấu nhắc của MATLAB trong student MATLAB. Trong các version khác của MATLAB dấu nhắc đơn giản chỉ là ">>". Khi cửa sổ lệnh xuất hiện, là cửa sổ hoạt động, con trỏ xuất hiện bên phải dấu nhắc như ở hình dưới. Con trỏ và dấu nhắc nay của MATLAB báo rằng MATLAB đang đợi để thực hiện lệnh.

1.1 Các phép toán đơn giản

Giống như máy tính đơn giản thông thường, MATLAB có thể thực hiện các phép toán đơn giản, ví dụ 1:

Hải đến một cửa hàng văn phòng phẩm và mua 4 cục tẩy, 25 xu một cục, 6 tập vở, 52 xu một tập, hai cuộn băng cassette 99 xu một cuộn. Hãy tính xem Hải mua bao nhiêu vật, và tổng số tiền là bao nhiêu?

Nếu dùng máy tính thông thường, ta vào các số:

$$4 + 6 + 2 = 12 \text{ (vật)}$$

$$4 \times 25 + 6 \times 52 + 2 \times 99 = 610 \text{ (xu)}$$

Trong MATLAB chúng ta có thể giải quyết vấn đề này theo nhiều cách. Trước tiên giống như một máy tính đơn giản, chúng ta có thể tính:

```
>> 4 + 6 + 2
```

```
ans=
```

```
12
```

```
>> 4*25 + 6*52 + 2*99
```

```
ans=
```

```
610
```

Chú ý rằng MATLAB không chú ý đến những khoảng trống, cho tất cả các phần, và phép nhân có mức độ ưu tiên cao hơn phép cộng. Và một chú ý khác là MATLAB gọi kết quả ans (viết tắt của answer) cho cả hai phép tính.

Như đã nói ở trên, vấn đề trên có thể giải quyết bằng cách chứa các thông tin vào biến của MATLAB:

```
>> erasers = 4
```

```

erasers=
    4
>> pads = 6
pads=
    6
>> tape = 2;
>> itirms = erases + pads + tape
itirms=
    12
>> cost = erases*25 + pads*52 + tape*99
cost=
    610

```

Ở đây chúng ta tạo 3 biến MATLAB: erases, pads, tape để chứa số lượng mỗi loại vật. Sau khi vào các giá trị cho các biến này, MATLAB hiển thị kết quả ra màn hình, trừ trường hợp biến tape. Dấu chấm phẩy đằng sau câu lệnh ">> tape = 2;" thông báo cho MATLAB nhận giá trị gán nhưng không hiển thị ra màn hình. Cuối cùng khác với gọi kết quả ans, chúng ta yêu cầu MATLAB gọi kết quả tổng số các vật là itirms, và tổng số tiền là cost. Tại mỗi bước MATLAB đều đưa ra các thông tin. Vì có lưu giữ các biến nên chúng ta có thể yêu cầu MATLAB tính giá trị trung bình cho mỗi vật:

```

>> everage_cost = cost/itirms
everage_cost=
    50.8333

```

Bởi vì everage cost có hai từ, mà MATLAB yêu cầu biến chỉ có một từ, nên chúng ta dùng dấu gạch dưới để nối hai từ này thành một từ.

Ngoài các phép tính trên, MATLAB còn có một số phép tính cơ bản khác như bảng dưới đây:

Phép tính	Ký hiệu	Ví dụ
Phép cộng, $a + b$	+	$5 + 3$
Phép trừ, $a - b$	-	$7 - 4$
Phép nhân, $a.b$	*	$18*24$
Phép chia, $a : b$	/ hoặc \	$56/8 = 8$ hoặc $56 \setminus 8$

Phép lũy thừa, a^b	$^$	5^2
----------------------	-----	-------

Trong các phép toán trên có mức độ ưu tiên khác nhau. khi tính từ trái sang phải của một dòng gồm nhiều lệnh thì phép toán lũy thừa có mức độ ưu tiên cao nhất, tiếp theo là phép nhân và phép chia có mức độ ưu tiên bằng nhau cuối cùng là phép cộng và phép trừ cũng có mức độ ưu tiên bằng nhau.

1.2 Không gian làm việc của MATLAB

Cũng như bạn làm việc với cửa sổ Lệnh, MATLAB nhớ các lệnh bạn gõ vào cũng như các giá trị bạn gán cho nó hoặc nó được tạo lên. Những lệnh và biến này được gọi là lưu giữ trong không gian làm việc của MATLAB, và có thể được gọi lại khi bạn muốn. Ví dụ, để kiểm tra giá trị của biến tape, tất cả những gì bạn phải làm là yêu cầu MATLAB cho biết bằng cách đánh vào tên biến tại dấu nhắc:

```
>> tape
tape=
     2
```

Nếu bạn không nhớ tên biến, bạn có thể yêu cầu MATLAB cho danh sách các biến bằng cách đánh lệnh *who* từ dấu nhắc lệnh:

```
>> who
Your variables are:
ans cost items tape
average_cost erasers pads
```

Chú ý rằng MATLAB không đưa ra giá trị của tất cả các biến, nếu bạn muốn biết giá trị, bạn đánh vào tên biến tại dấu nhắc lệnh của MATLAB.

Để gọi lại các lệnh bạn đã dùng, MATLAB dùng các phím mũi tên ($\leftarrow \rightarrow \uparrow \downarrow$) trên bàn phím của bạn. Ví dụ để gọi lại lệnh bạn gõ vào lúc gần hiện tại nhất, bạn nhấn phím mũi tên \uparrow , tiếp tục nhấn phím này, nó sẽ lại gọi tiếp lệnh trước đó. Nếu bạn dùng phím mũi tên \downarrow nó sẽ gọi lại lệnh từ lệnh đầu tiên cho đến lệnh gần hiện tại nhất. Các phím mũi tên \leftarrow và \rightarrow có thể dùng để thay đổi vị trí con trỏ trong dòng lệnh tại dấu nhắc của MATLAB, như vậy chúng ta có thể sửa dòng lệnh, thêm nữa, chúng ta có thể dùng chuột cùng với bộ nhớ đệm để cắt, copy, dán, và sửa văn bản tại dấu nhắc của dòng lệnh.

1.3 Biến

Giống như những ngôn ngữ lập trình khác, MATLAB có những quy định riêng về tên biến. Trước tiên tên biến phải là một từ, không chứa dấu cách, và tên biến phải có những quy tắc như những quy tắc sau:

Quy định về tên biến	Chỉ dẫn/ Ví dụ
Tên biến có phân biệt chữ hoa chữ thường	ltems, items, itErms, và lTERMS là các biến khác nhau
Tên biến có thể chứa nhiều nhất 31 kí tự, còn các kí tự sau kí tự thứ 31 bị bỏ đi	howaboutthisveriablename
Tên biến bắt đầu phải là chữ cái, tiếp theo có thể là chữ số, số gạch dưới	how_about_this_variable_name, X51483, a_b_c_d_e
Kí tự chấm câu không được phép dùng vì nó có những ý nghĩa đặc biệt	

Cùng với những quy định trên, MATLAB có những biến đặc biệt trong bảng sau:

Các biến đặc biệt	Giá trị
ans	Tên biến mặc định dùng để trả về kết quả
pi	$\pi = 3.1415..$
eps	Số nhỏ nhất, như vậy dùng cộng với 1 để được số nhỏ nhất lớn hơn 1
flops	Số của phép toán số thực
inf	Để chỉ số vô cùng như kết quả của 1/0
NaN hoặc nan	Dùng để chỉ số không xác định như kết quả của 0/0
i (và) j	$i = j = \sqrt{-1}$
nargin	Số các đối số đưa vào hàm được sử dụng
narout	Số các đối số hàm đưa ra
realmin	Số nhỏ nhất có thể được của số thực
realmax	Số lớn nhất có thể được của số thực

Như bạn có thể tạo một biến của MATLAB, và bạn cũng có thể gán lại giá trị cho một hoặc nhiều biến. Ví dụ:

```
>> erases = 4;

>> pads = 6;

>> tape = 2;

>> items = erases + pads + tape

items=

    12

>> erases = 6

erases=

     6

>> items

items=

    12
```

Ở đây chúng ta sử dụng lại ví dụ trên, chúng ta tìm được số vật mà Hải đã mua sau đó chúng ta thay đổi số cục tẩy lên 6, giá trị này sẽ đè lên giá trị trước của nó là 4. Khi bạn làm như vậy, giá trị của `items` vẫn không thay đổi, vì MATLAB không tính lại `items` với giá trị mới của `erases`. Khi MATLAB thực hiện một phép tính, nó lấy giá trị của các biến hiện thời, nên nếu bạn muốn tính giá trị mới của `items`, `cost`, `average_cost`, bạn gọi lại các lệnh tính các giá trị đó.

Đối với các biến đặc biệt ở trên, nó có sẵn giá trị, như vậy khi bạn khởi động MATLAB, nếu bạn thay đổi giá trị của nó thì những giá trị đặc biệt ban đầu sẽ bị mất cho đến khi bạn xóa biến đó đi hoặc khởi động lại MATLAB. Do đó bạn không nên thay đổi giá trị của biến đặc biệt, trừ khi nó thực sự cần thiết.

Các biến trong không gian làm việc của MATLAB có thể bị xóa không điều kiện bằng cách dùng lệnh ***clear***. Ví dụ:

```
>> clear erases

    chỉ xóa một biến erases

>> clear cost items

    xóa cả hai biến cost và items

>> clear *
```

dấu * để chỉ rằng xóa tất cả các biến bắt đầu bằng hai ký tự cl.


```
>> clear
```

xoá tất cả các biến trong không gian làm việc!. Bạn sẽ không được hỏi để xác nhận câu lệnh này và tất cả các biến đã bị xoá không thể khôi phục lại.

Có thể nói rằng dùng lệnh *clear* rất nguy hiểm, vì vậy khi dùng lệnh này bạn phải rất thận trọng.

1.4 Câu giải thích (comment) và sự chấm câu

Tất cả các văn bản đứng sau kí hiệu phần trăm (%) đều là câu giả thích. Ví dụ:

```
>> erases = 4 % Số cục tẩy.
```

```
erases=
```

```
4
```

Biến *erases* được gán giá trị là 4, còn tất cả kí hiệu phần trăm và văn bản đứng sau nó đều bị bỏ đi. Quy ước này giúp cho chúng ta dễ theo dõi công việc chúng ta đang làm.

Nhiều lệnh có thể đặt trên cùng một hàng, chúng cách nhau bởi dấu phẩy hoặc dấu chấm phẩy, như:

```
>> erases = 4; pads = 6; tape = 2
```

```
erases=
```

```
4
```

```
tape=
```

```
2
```

Dấu phẩy để yêu cầu MATLAB hiển thị kết quả trên màn hình; còn dấu chấm phẩy là không hiển thị kết quả trên màn hình.

```
>> average_cost = cost/ ...
```

```
iterns
```

```
average_cost=
```

```
50.83333
```

Như ví dụ trên, ta có thể dùng dấu ba chấm (...) để chỉ câu lệnh được tiếp tục ở hàng dưới, phép tính thực hiện được khi dấu ba chấm ngăn cách giữa toán tử và biến, nghĩa là tên biến không bị ngăn cách giữa hai hàng:

```
>> average_cost = cost/ it...
```

```
erms
```

```
??? 'age_cost = cost/terms
```

```
|
```

Missing operator, coma, or semicolon.

giống như vậy, trạng thái của lời giải thích không thể tiếp tục:

```
>> % Comments cannot be continued ...
```

```
>> either
```

```
??? Undefined function or variable either.
```

Bạn có thể dùng chương trình bằng cách nhấn đồng thời Ctrl và C.

1.5 Số phức

Một trong những ưu thế của MATLAB là làm việc với số phức. Số phức trong MATLAB được định nghĩa theo nhiều cách, ví dụ như sau:

```
>> c1 = 1 - 2i % Chèn thêm kí tự i vào phần ảo.
```

```
c1=
```

```
1.0000 - 2.0000i
```

```
>> c1 = 1 - 2j % j ở đây tương tự như i ở trên.
```

```
c1=
```

```
1.0000 - 2.0000i
```

```
>> c2 = 3*(2-sqrt(-1)*3)
```

```
c2=
```

```
6.0000 - 9.0000i
```

```
>> c3 = sqrt(-2)
```

```
c3=
```

```
0 + 1.4142i
```

```
>> c4 = 6 + sin(.5)*i
```

```
c4=
```

```
6.0000 + 0.4794i
```

```
>> c5 = 6 + sin(.5)*j
```

```
c5=
```

$$6.0000 + 0.4794i$$

Trong hai ví dụ cuối, MATLAB mặc định giá trị của $i = j = \sqrt{-1}$ dùng cho phần ảo. Nhân với i hoặc j được yêu cầu trong trường hợp này, $\sin(.5)i$ và $\sin(.5)j$ không có ý nghĩa đối với MATLAB. Cuối cùng với các kí tự i và j , như ở trong hai ví dụ đầu ở trên chỉ làm việc với số cố định, không làm việc được với biểu thức.

Một số ngôn ngữ yêu cầu sự điều khiển đặc biệt cho số phức khi nó xuất hiện, trong MATLAB thì không cần như vậy. Tất cả các phép tính đều thao tác được như đối với số thực thông thường:

```
>> c6 = (c1 + c2)/c3 % Từ các dữ liệu ở trên
```

```
c6=
```

$$-7.7782 - 4.9497i$$

```
>> check_it_out = i^2 % Bình phương của i phải là -1
```

```
check_it_out=
```

$$-1.0000 + 0.0000i$$

trong ví dụ này chỉ còn lại phần thực, phần ảo bằng không. Chúng ta có thể dùng hàm *real* và *imag* để kiểm tra từng phần thực và ảo.

Chúng ta có thể biểu diễn số phức dạng cực (độ lớn và góc):

$$M\angle\theta = M.e^{j\theta} = a+bi$$

Ở trên số phức được biểu diễn bằng độ lớn M và góc θ , quan hệ giữa các đại lượng này và phần thực, phần ảo của số phức biểu diễn dưới dạng đại số là:

$$M = \sqrt{a^2 + b^2}$$

$$\theta = \tan^{-1}(b/a)$$

$$a = M\cos\theta$$

$$b = M\sin\theta$$

Trong MATLAB, để chuyển từ dạng cực sang dạng đại số, dùng các hàm *real*, *imag* và *angle*.

```
>> c1 % Gọi lại c1
```

```
c1=
```

$$1.0000 - 2.0000i$$

```
>> M_c1 = abs(c1) % Tính argument của số phức
```

```

M_c1=
    2.2361
>> angle_c1 = angle(c1)      % Tính góc của số phức theo radian
angle_c1=
    -1.1071
>> deg_c1 = angle_c1*180/ pi    % Chuyển từ radian sang độ
    -63.4349
>> real_c1 = real(c1)          % Tính phần thực
real_c1=
    1
>> imag_c1 = imag(c1)         % Tính phần ảo
imag_c1=
    -2

```

CÁC BÀI TOÁN KỸ THUẬT

Tương tự như hầu hết các máy tính kỹ thuật, MATLAB có thể đưa ra rất nhiều các hàm toán học, kỹ thuật thông dụng, ngoài ra MATLAB còn cung cấp hàng trăm các hàm đặc biệt và thuật toán, nó rất hữu ích để giải quyết các vấn đề khoa học. Tất cả các hàm này được liệt kê trong online help, ở đây chỉ đề cập đến những hàm thông dụng nhất.

2.1 Các hàm toán học thông thường

Các hàm toán học của MATLAB được liệt kê trong bảng các hàm, chúng đều có chung một cách gọi hàm như ví dụ sau:

```
>> x = sqrt(2)/2
```

```
x=
```

```
0.7071
```

```
>> y = sin(x)
```

```
y=
```

```
0.7854
```

```
>> y_deg = y*180/pi
```

```
y_deg=
```

```
45.0000
```

Những lệnh này để tìm một góc (tính bằng độ) khi biết giá trị hàm sin của nó là $\sqrt{2} / 2$.

Tất cả các hàm liên quan đến góc của MATLAB đều làm việc với radian.

Các hàm thông thường	
<code>abs(x)</code>	Tính argument của số phức x
<code>acos(x)</code>	Hàm ngược của cosine
<code>acosh(x)</code>	Hàm ngược của hyperbolic cosine
<code>angle(x)</code>	Tính góc của số phức x
<code>asin(x)</code>	Hàm ngược của sine
<code>asinh(x)</code>	Hàm ngược của hyperbolic sine
<code>atan(x)</code>	Hàm ngược của tangent
<code>atan2(x, y)</code>	Là hàm arctangent của phần thực của x và y
<code>atanh(x)</code>	Hàm ngược của hyperbolic tangent
<code>ceil(x)</code>	Xấp xỉ dương vô cùng
<code>conj(x)</code>	Số phức liên hợp
<code>cos(x)</code>	Hàm cosine của x
<code>cosh(x)</code>	Hàm hyperbolic cosine của x
<code>exp(x)</code>	Hàm e^x
<code>fix(x)</code>	Xấp xỉ không
<code>floor(x)</code>	Xấp xỉ âm vô cùng
<code>gcd(x, y)</code>	Ước số chung lớn nhất của hai số nguyên x và y
<code>imag(x)</code>	Hàm trả về phần ảo của số phức
<code>lcm(x, y)</code>	Bội số chung nhỏ nhất của hai số nguyên x và y
<code>log(x)</code>	Logarithm tự nhiên
<code>log10(x)</code>	Logarithm cơ số 10
<code>real(x)</code>	Hàm trả về phần thực của x
<code>rem(x, y)</code>	Phần dư của phép chia x/ y
<code>round(x)</code>	Hàm làm tròn về số nguyên tố
<code>sign(x)</code>	Hàm dấu: trả về dấu của argument:
<code>sin(x)</code>	Hàm tính sine của x
<code>sinh(x)</code>	Hàm tính hyperbolic sine của x
<code>sqrt(x)</code>	Hàm khai căn bậc hai
<code>tan(x)</code>	Tangent
<code>tanh(x)</code>	Hyperbolic tangent

```
>> 4*atan(1)      % Một cách tính xấp xỉ giá trị của pi
```

```
ans=
```

```
3.1416
```

```
>> help atan2      % Yêu cầu giúp đỡ đối với hàm atan2
```

```
ATAN2 four quadrant inverse tangent
```

ATAN2(Y, X) is the four quadrant arctangent of the real parts of the elements of X and Y. -
pi <= ATAN2(Y, X) <= pi

see also ATAN.

```
>> 180/pi*atan(-2/ 3)
```

```
ans=
```

```
-33.69
```

```
>> 180/pi*atan2(2, -3)
```

```
ans=
```

```
146.31
```

```
>> 180/pi*atan2(-2, 3)
```

```
ans=
```

```
-33.69
```

```
>> 180/pi*atan2(2, 3)
```

```
ans=
```

```
33.69
```

```
>> 180/pi*atan2(-2, -3)
```

```
ans=
```

```
-146.31
```

Một số ví dụ khác:

```
>> y = sqrt(3^2 + 4^2)      % Tính cạnh huyền của tam giác pitago 3-4-5
```

```
y=
```

```
5
```

```
>> y = rem(23,4)           % 23/4 có phần dư là 3
```

```

y=
    3
>> x = 2.6,y1 = fix(x),y2 = floor(x),y3 = ceil(x),y4 = round(x)
x=
    2.6000
y1=
    2
y2=
    2
y3=
    3
y4=
    3
>> gcd(18,81)    % 9 là ước số chung lớn nhất của 18 và 81
ans=
    9
>> lcm(18,81)    % 162 là bội số chung lớn nhất của 18 và 81
ans=
    162

```

2.2. Các ví dụ

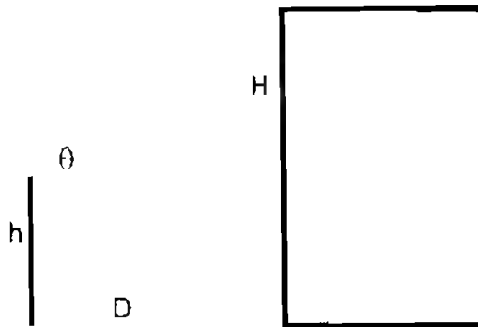
Ví dụ: Ước lượng chiều cao của ngôi nhà

Vấn đề: Giả thiết biết khoảng cách từ người quan sát đến ngôi nhà là D , góc từ người quan sát đến ngôi nhà là θ ; chiều cao của người quan sát là h . Hỏi ngôi nhà cao bao nhiêu?

Giải pháp: Ta biểu diễn kích thước như hình 2.1:

Ngôi nhà có chiều cao là $H + h$, H là chiều dài của một cạnh của tam giác, chiều dài này có thể tính được bằng công thức quan hệ giữa góc và cạnh của tam giác:

$$\tan(\theta) = \frac{H}{D}$$



Hình 2.1

Từ đó ta có chiều cao của ngôi nhà là

$$h + H = h + D \cdot \tan(\theta)$$

Nếu $h = 2\text{m}$, $D = 50\text{m}$, và θ là 60° , MATLAB sẽ đưa ra kết quả là:

```
>> h = 2
h =
    2
>> theta = 60
theta =
    60
>> D = 50
D =
    50
>> buiding_height = h+D*atan(theta*pi/180)
buiding_height =
    54.3599
```

Ví dụ sự suy giảm do phân rã

Vấn đề Sự phân rã phân tử polonium có chu kỳ phân rã là 140 ngày, tức là sau 140 ngày thì lượng polonium còn lại là $1/2$ lượng ban đầu. Hỏi nếu ban đầu có 10 grams polonium, nó sẽ còn lại bao nhiêu sau 250 ngày?

Giải quyết: Sau 1 chu kỳ phân rã hoặc 140 ngày, còn lại $10 \times 0.5 = 5$ grams; sau 2 chu kỳ phân rã hoặc 280 ngày, còn lại $5 \times 0.5 = 10 \times (0.5)^2 = 2.5$ grams, từ đó ta có kết quả nằm trong khoảng 5 và 2.5 grams, và ta có công thức tính phần còn lại sau khoảng thời gian bất kỳ:

khối lượng còn lại = khối lượng ban đầu $\times (0.5)^{(\text{thời gian} / \text{chu kỳ})}$

ví dụ thời gian là 250 ngày, và kết quả MATLAB đưa ra là:

```
>> initial_amount = 10; % Khối lượng ban đầu
```

```
>> half_life = 140; % Chu kỳ phân rã
```

```
>> time = 250; % Thời gian tính khối lượng
```

```
>> amount_left = initial_*0.5^(time/half_life)
```

```
amount_left=
```

```
2.9003
```

Ví dụ tính toán về lãi xuất

Vấn đề: Bạn Kiểm và Dũng đồng ý mua ô tô mới với giá 18,500 dollars. Người bán ô tô đưa ra hai giải pháp về tài chính là: thứ nhất, trả 2.9% lãi suất của số tiền trên trong vòng 4 năm. Thứ hai là trả 8.9% lãi suất năm của số tiền trên trong vòng 4 năm và giá bán được giảm đi một khoản là 1500 dollars. Hỏi với giải pháp nào thì bạn Kiểm và Dũng mua được ô tô với giá rẻ hơn?

Giải pháp: Số tiền trả hàng tháng là P, trên tổng số tiền là A dollars, tỉ số lãi suất hàng tháng là R, trả trong M tháng:

$$P = A \left[\frac{R(1+R)^M}{(1+R)^M - 1} \right]$$

Tổng số tiền phải trả sẽ là: $T = P \times M$

Giải pháp MATLAB đưa ra là:

```
>> format bank % Dùng dạng hiển thị ngân hàng
```

```
>> A = 18500; % Tổng số tiền
```

```
>> M = 12*4; % Số tháng phải trả lãi
```

```
>> FR = 1500; % Tiền giảm giá của nhà máy
```

```
>> % Giải pháp thứ nhất
```

```
>> R = (2.9/100)/12; % Tỷ lệ lãi suất hàng tháng
```

```
>> P = A*(R*(1+R)^M/((1+R)^M - 1)) % Khoản tiền phải trả hàng tháng
```

P=

408.67

>> T1 = P*M % Tổng giá trị của ô tô

T1=

19616.06

>> % Giải pháp thứ hai

>> R = (8.9/100)/12; % Tỷ lệ lãi suất hàng tháng

>> P = (A-FR)*(R*(1 + R)^M/((1+R)^M - 1)) % Tiền phải trả hàng tháng

P=

422.24

>> T2 = P*M % Tổng giá trị của ô tô

T2=

20267.47

>> Diff = T2 - T1

Diff=

651.41

Như vậy ta có giải pháp thứ nhất giá rẻ hơn giải pháp thứ hai.

Ví dụ: Vấn đề nồng độ axit

Vấn đề: Như một phần của quy trình sản xuất một chi tiết theo phương pháp đúc tại một nhà máy tự động, chi tiết đó được nhúng trong nước để làm nguội, sau đó nhúng trong bồn đựng dung dịch axit để làm sạch. Trong toàn bộ của quá trình nồng độ axit giảm đi khi các bộ phận được lấy ra khỏi bồn axit vì khi nhúng bộ phận của vật đúc vào bồn thì một lượng nước còn bám trên vật đúc khi nhúng ở bể trước cũng vào theo và khi nhấc ra khỏi bồn một lượng axit bám theo vật. Để đảm bảo chất lượng thì nồng độ axit phải không được nhỏ hơn một lượng tối thiểu. Bạn hãy bắt đầu với nồng độ dung dịch là 90% thì nồng độ tối thiểu phải là 50%. Lượng chất lỏng thêm vào và lấy đi sau mỗi lần nhúng dao động trong khoảng từ 1% đến 10%. Hỏi bao nhiêu chi tiết có thể nhúng vào bể dung dịch axit trước khi nồng độ của nó giảm xuống dưới mức cho phép?

Giải pháp:

Ban đầu nồng độ axit là $initial_con = 90\% = \text{axit} / (\text{axit} + \text{water})$

sau lần nhúng thứ nhất nồng độ axit còn:

$$\begin{aligned}
 \text{con} &= \frac{\text{acid}}{(\text{acid} + \text{water}) - \text{wateradded}} \\
 &= \frac{\text{acid}}{(\text{acid} + \text{water}) + \text{lost}(\text{acid} + \text{water})} \\
 &= \frac{\text{acid}}{(1 + \text{lost})(\text{acid} + \text{water})} \\
 &= \frac{\text{initial_con}}{(1 + \text{lost})}
 \end{aligned}$$

"acid" là lượng axit ban đầu trong dung dịch, "water" là lượng nước ban đầu trong dung dịch, "lost" là lượng phần trăm nước thêm vào. Số axit còn lại trong dung dịch sau lần nhúng thứ nhất là:

$$\text{acid_left} = \frac{\text{acid}}{(1 + \text{lost})}$$

Nghĩa là, khi nhúng lần thứ hai nồng độ dung dịch sẽ là:

$$\begin{aligned}
 \text{con} &= \frac{\text{acid_left}}{(\text{acid_water}) + \text{wateradded}} \\
 &= \frac{\text{acid_left}}{(1 + \text{lost})(\text{acid} + \text{water})} \\
 &= \frac{\text{initial_con}}{(1 + \text{lost})^2}
 \end{aligned}$$

Tiếp tục quá trình này, sau n lần nhúng, nồng độ axit là:

$$\text{con} = \frac{\text{initial_con}}{(1 + \text{lost})^n}$$

Nếu nồng độ axit còn lại là mức tối thiểu chấp nhận được, số lần nhúng cực đại sẽ là một số nguyên bằng hoặc nhỏ hơn n:

$$n = \frac{\log(\text{initial_con} / \text{min_con})}{\log(1 + \text{lost})}$$

Trong MATLAB giải pháp sẽ là:

```
>> initial_con = 90
```

```
initial_con =
```

```

90
>> min_con = 50
min_con=
50
>> lost = 0.01;
>> n = floor(log(initial_con/min_con)/log(1+lost))
n=
59

```

Như vậy có thể nhúng 59 lần trước khi nồng độ axit giảm xuống dưới 50%. Chú ý hàm *floor* dùng để làm tròn số n xuống số nguyên gần nhất, và ở đây ta cũng có thể dùng hàm *logarithm* cơ số 10 và *logarithm* cơ số 2 thay cho hàm *logarithm* tự nhiên ở trên.

NHỮNG ĐẶC ĐIỂM CỦA CỬA SỔ LỆNH TRONG MATLAB

Cửa sổ lệnh (command) của MATLAB có rất nhiều những đặc điểm cần chú ý một số chúng đã được giới thiệu ở chương trước, và sau đây chúng ta tìm hiểu rõ hơn về chúng.

3.1 Quản lí không gian làm việc của MATLAB

Các dữ liệu và biến được tạo lên trong cửa sổ lệnh, được lưu trong một phần gọi là không gian làm việc của MATLAB. Muốn xem tên biến trong không gian làm việc của MATLAB ta dùng lệnh who:

```
>> who
```

Your variables are:

```
D      h
```

```
buiding_height      theta
```

Các biến này được dùng trong ví dụ ước lượng chiều cao ngôi nhà. Để xem chi tiết hơn về các biến ta dùng lệnh whos:

```
>> whos
```

Name	Size	Bytes	Class
D	1x1 8	double	array
buiding_height	1x1 8	double	array
h	1x1 8	double	array
theta	1x1 8	double	array

Grand total is 4 elements using 32 bytes

Mỗi biến được liệt kê với kích cỡ của nó, số bytes sử dụng và các lớp của chúng (class). trong ví dụ đặc biệt này, các biến đều là số đơn, có độ chính xác hai số sau dấu phẩy. Lệnh *whos* đặc biệt có ích khi nghiên cứu đến phần mảng và các kiểu dữ liệu khác.

Ngoài các hàm này, trong mục **Show Workspace** trong bảng chọn **file** tạo ra cửa sổ GUI gọi là **Workspace Browser**, nó chứa các thông tin tương tự như lệnh *whos*. Thêm nữa nó tạo cho bạn khả năng xoá, làm sạch các biến mà bạn chọn. Cửa sổ này cũng có thể tạo bằng cách nhấn nút **Workspace Browser**, trên thanh công cụ của cửa sổ lệnh.

Như đã trình bày ở trên, lệnh *clear* có thể xoá biến từ không gian làm việc của MATLAB. Ví dụ:

```
>> clear h D % Xoá các biến h và D
```

```
>> who
```

Your variables are:

buiding_height theta

Các tùy chọn khác của hàm *clear* chúng ta có thể tìm hiểu thêm bằng lệnh *help*:

```
>> help clear
```

CLEAR Clear variables and functions from memory.

CLEAR removes all variables from the workspace.

CLEAR VARIABLES does the same thing.

CLEAR GLOBAL removes all global variables.

CLEAR FUNCTIONS removes all compiled M-functions.

CLEAR MEX removes all links to MEX-files.

CLEAR ALL removes all variables, globals, functions and MEX links.

CLEAR VAR1 VAR2 ... clears the variables specified. The wildcard character '*' can be used to clear variables that match a pattern.

For instance, CLEAR X* clears all the variables in the current workspace that start with X.

If X is global, CLEAR X removes X from the current workspace, but leaves it accessible to any functions declaring it global.

CLEAR GLOBAL X completely removes the global variable X.

CLEAR FUN clears the function specified. If FUN has been locked by MLOCK it will remain in memory.

CLEAR ALL also has the side effect of removing all debugging breakpoints since the breakpoints for a file are cleared whenever the m-file changes or is cleared.

Use the functional form of CLEAR, such as CLEAR('name'), when the variable name or function name is stored in a xâu.

See also WHO, WHOS, MLOCK, MUNLOCK.

Cuối cùng, khi làm việc trong không gian làm việc của MATLAB, có thể dễ dàng ghi hoặc in một bản sao công việc của bạn, lệnh *diary* ghi dữ liệu người dùng đưa vào và cửa sổ lệnh và đưa ra file văn bản dạng mã ASCII có tên là diary trong thư mục hiện tại:

```
>> diary frame % ghi dữ liệu vào file frame
```

```
>> diary off % kết thúc lệnh diary và đóng file
```

Khi cửa sổ lệnh được chọn, chọn **print...** từ bảng chọn **file** để in một bản của cửa sổ lệnh, bạn có thể dùng chuột để lựa chọn phần mình muốn ghi, chọn **Print Selection...** từ bảng chọn **file**, để in một phần văn bản đã lựa chọn.

3.2 Ghi và phục hồi dữ liệu

Để nhớ các biến, MATLAB có thể ghi và gọi lại dữ liệu từ file trong máy tính của bạn. Mục **Workspace as...** trong bảng chọn **file** mở hộp chuẩn hội thoại để ghi tất cả các biến hiện tại. Giống như vậy, trong mục **Load Workspace** trong bảng chọn **file** mở hộp hội thoại để gọi lại tất cả các biến mà ta đã ghi lại từ không gian làm việc trước, nó không làm mất các biến này trong không gian làm việc hiện tại. Khi ta gọi lại các biến, mà các biến này trùng tên với các biến trong không gian làm việc của MATLAB, nó sẽ thay đổi giá trị của các biến theo giá trị của các biến gọi ra từ file.

Nếu bảng chọn **file** không thuận tiện hoặc không đáp ứng được những yêu cầu của bạn, MATLAB cung cấp hai lệnh *save* và *load*, nó thực hiện một cách mềm dẻo hơn, trong trường hợp đặc biệt, lệnh *save* cho phép bạn ghi một hoặc nhiều hơn một biến tùy theo sự lựa chọn của bạn.

Ví dụ:

```
>> save
```

Chứa tất cả các biến trong MATLAB theo kiểu nhị phân trong file MATLAB.mat

>> save data

chứa tất cả các biến trong MATLAB theo kiểu nhị phân trong file data.mat

>> save data erasers pads tape -ascii

Ghi các biến erasers, pads, tape trong dạng mã ASCII 8 số trong file data. File dạng mã ASCII có thể sửa đổi bằng bất cứ chương trình soạn thảo văn bản nào, chú ý rằng file ASCII không có phần mở rộng .mat.

>> save data erasers pads tape -ascii -double

Ghi các biến erasers, pads, tape dạng ASCII 16 số trong file data.

Lệnh **load** cũng dùng với cú pháp tương tự.

3.3 Khuôn dạng hiển thị số

Khi MATLAB hiển thị kết quả dạng số, nó tuân theo một số quy định sau:

Mặc định, nếu kết quả là số nguyên thì MATLAB hiển thị nó là một số nguyên, khi kết quả là một số thực thì MATLAB hiển thị số xấp xỉ với bốn chữ số sau dấu phẩy, còn các số dạng khoa học thì MATLAB hiển thị cũng giống như trong các máy tính khoa học.

Bạn có thể không dùng dạng mặc định, mà tạo một khuôn dạng riêng từ mục **Preferences**, trong bảng chọn **file**, có thể sang trang mặc định hoặc đánh dạng xấp xỉ tại dấu nhắc.

Chúng ta dùng biến `average_cost` (trong ví dụ trước) làm ví dụ, dạng số này là:

Lệnh của MATLAB	Average_cost	Chú thích
format short	50.833	5 số
format long	50.83333333333334	16 số
format short e	5.0833e+01	5 số với số mũ
format long e	5.08333333333334e+01	16 số với số mũ
format short g	50.833	chính xác hơn format short hoặc format short e
format long g	50.83333333333333	chính xác hơn format long hoặc format long e
format hex	40496aaaaaaaaaab	hệ cơ số 16
format bank	50.83	hai số hệ 10
format +	+	dương, âm hoặc bằng không

format rat	3/5/6	dạng phân số
------------	-------	--------------

Một chú ý quan trọng là MATLAB không thay đổi số khi định lại khuôn dạng hiển thị được chọn, mà chỉ thay đổi màn hình thay đổi.

SCRIPT M_FILES

Trong MatLab, yêu cầu của bạn tại dấu nhắc của MATLAB trong cửa sổ lệnh là nhanh và hiệu quả. Tuy nhiên vì số lệnh tăng lên, hoặc khi bạn muốn thay đổi giá trị của một hoặc nhiều biến và thực hiện lại một số lệnh với giá trị mới, nếu cứ đánh lặp lại tại dấu nhắc của MATLAB thì sẽ trở nên buồn tẻ, do vậy MATLAB cung cấp một giải pháp cho vấn đề này là: nó cho phép bạn thay thế các lệnh của MATLAB bằng một file văn bản đơn giản, và yêu cầu MATLAB mở file và thực hiện lệnh chính xác như là đánh tại dấu nhắc của MATLAB tại cửa sổ lệnh, những file này gọi là **script file**, hoặc đơn giản là **M_file**. Danh từ "script" để chỉ rằng thực tế MATLAB đọc từ file kịch bản tìm thấy trong file. Danh từ "M_file" để chỉ rằng tên script file đó phải kết thúc bằng phần mở rộng là '.m' như ví dụ example1.m.

Để tạo một script M_file, chọn **New** trong bảng chọn **file** và chọn **M_file**. Thủ tục này sẽ tạo ra màn hình soạn thảo, và bạn có thể đánh được các lệnh của MATLAB trong đó. Ví dụ dưới đây là cách lệnh trong ví dụ ước lượng chiều cao ngôi nhà ở trước:

```
function example1
    % example1.m Ví dụ ước lượng chiều cao ngôi nhà
    h = 2
    theta = 60
    D = 50;
    building_height = h + D*tan(theta*pi/180)
```

Bạn có thể ghi và lưu giữ file này bằng cách chọn **Save** từ bảng chọn file. Khi bạn ghi tên file chú ý phải đánh tên file trùng với tên hàm (example) không cần đánh vào phần mở rộng, MATLAB tự gán vào cho nó. Khi đó từ dấu nhắc ta có thể đánh:

```
>> example1
h=
    2
```

```
theta=
    60
building_height=
    54.3599
```

Khi MATLAB diễn giải các trạng thái của example1 ở trên, nó sẽ được nói kỹ hơn ở chương sau, nhưng một cách ngắn gọn, MATLAB dùng các trạng thái của biến MATLAB hiện tại và tạo lên các lệnh của nó, bắt đầu bằng tên M_file. Nghĩa là, nếu example1 không phải là biến hiện tại, hoặc một lệnh MATLAB xây dựng lên, MATLAB mở file example1.m (nếu nó tìm thấy) và tính giá trị các lệnh tìm thấy chỉ khi chúng ta vào các thông số chính xác tại dấu nhắc của cửa sổ lệnh. Như đã thấy lệnh trong M_file truy cập đến tất cả các biến trong không gian làm việc của MATLAB, và tất cả các biến trong M_file trở thành một phần của không gian làm việc. Bình thường các lệnh đọc trong M_file không được hiển thị như là nó được tính trong cửa sổ lệnh, nhưng lệnh **echo on** yêu cầu MATLAB hiển thị hoặc lặp lại lệnh đối với cửa sổ lệnh như chúng ta đã đọc và tính. Tiếp theo bạn có thể đoán được lệnh **echo off** làm gì. Giống như vậy, lệnh **echo** lặp lại bởi nó làm thay đổi chính trạng thái của nó.

Với đặc điểm này của M_file bạn có thể thay đổi lại nội dung của file, ví dụ bạn có thể mở M_file example1.m thay đổi lại các giá trị của h, D, hoặc theta, ghi lại file đó và yêu cầu MATLAB tính lại lệnh trong file. Thêm nữa, bằng cách tạo M_file, các lệnh của bạn được lưu trên đĩa và có thể sử dụng về sau khi bạn cần.

Những ứng dụng của chỉ dẫn của MATLAB giúp chúng ta hiểu được khi dùng script file như trong example1.m, chỉ dẫn cho phép bạn lưu giữ cùng các lệnh trong script file, vì vậy bạn nhớ được những lệnh đó làm gì khi bạn nhìn lại file sau đây. Thêm nữa, dấu chấm phẩy đằng sau câu lệnh không cho hiển thị kết quả, từ đó bạn có thể điều chỉnh script file đưa ra những kết quả cần thiết.

Vì những ứng dụng của script file, MATLAB cung cấp một số hàm đặc biệt có ích khi bạn sử dụng trong M_file:

Các hàm M_file	
disp(ans)	Hiển thị các kết quả mà không hiện tên biến
Echo	Điều khiển cửa sổ lệnh lặp lại các lệnh của script file
Input	Sử dụng dấu nhắc để đưa dữ liệu vào
Keyboard	Trao điều khiển tạm thời cho bạn phím
Pause	Dừng lại cho đến khi người dùng nhấn một phím bất kỳ
pause(n)	Dừng lại n giây
Waitforbuttonpress	Dừng lại cho đến khi người dùng nhấn chuột hoặc phím.

Khi lệnh của MATLAB không kết thúc bằng dấu chấm phẩy, kết quả của lệnh được hiển thị trên cửa sổ lệnh cùng với tên biến. Đôi lúc nó không cho hiển tên biến, trong MATLAB ta dùng lệnh `disp` để thực hiện việc này

```
>> h           % Cách truyền thống để hiển kết quả
```

```
h=
    2
```

```
>> disp(h)     % Hiện kết quả không có tên biến
```

```
2
```

Để giúp bạn soạn thảo script file khi tính toán cho nhiều trường hợp, lệnh *input* cho phép bạn tạo câu nhắc để vào dữ liệu được an toàn. Ví dụ `example1.m` với những phần được sửa:

```
function example1
```

```
    % example1.m Ví dụ ước lượng chiều cao ngôi nhà
```

```
h = 2
```

```
theta = 60
```

```
D = input(' Vào khoảng cách giữa người và ngôi nhà: ')
```

```
building_height = h + D*tan(theta*pi/180)
```

```
chạy file này:
```

```
>> example1
```

```
n=
```

```
    2
```

```
theta=
```

```
    60
```

```
Vào khoảng cách giữa người và ngôi nhà: 60
```

```
D=
```

```
    60
```

```
building_height=
```

```
   64.8319
```

Ở ví dụ trên ta gõ vào số 60 và ấn Enter. Những lệnh sau đó sẽ tính với giá trị của `D` là 60. Chú ý rằng hàm *input* có thể dùng với các phép toán khác giống như đối với các hàm thông

thường khác, hàm *input* cũng chấp nhận đối với bất cứ kiểu biểu diễn số nào, ví dụ ta vào một số là: $\sqrt{1908} + 5$.

```
>> example1
```

```
h=
```

```
2
```

```
theta=
```

```
60
```

Vào khoảng cách giữa người và ngôi nhà: $\text{sqrt}(1908)+5$

```
D=
```

```
48.6807
```

```
building_height=
```

```
52.9783
```

Để xem những tác động của lệnh *echo*, ta dùng chúng trong script file:

```
echo on
```

```
function example1
```

```
% example1.m Ví dụ ước lượng chiều cao ngôi nhà
```

```
h = 2
```

```
theta = 60
```

```
D = input(' Vào khoảng cách giữa người và ngôi nhà: ')
```

```
building_height = h + D*tan(theta*pi/180)
```

```
echo off
```

chạy chương trình ta được: _

```
>> example1
```

```
% example1.m Ví dụ ước lượng chiều cao ngôi nhà
```

```
h = 2
```

```
h=
```

```
2
```

```
theta = 60
```

```

theta=
    60
D = input(' Vào khoảng cách giữa người và ngôi nhà: ')
Vào khoảng cách giữa người và ngôi nhà: 60
building_height = h + D*tan(theta*pi/180)
building_height=
    64.8319
echo off

```

Như bạn đã thấy trong trường hợp này, lệnh *echo* làm cho kết quả khó đọc hơn, nhưng ngược lại lệnh nó có thể rất có ích khi gỡ rối nhiều script file ứng dụng.

QUẢN LÝ TỆP

MATLAB cung cấp một số các hàm file hệ thống và các lệnh cho phép bạn liệt kê tên file, xem, và xóa M_file, hiển thị và thay đổi thư mục chứa nó. Một số tổng kết các lệnh được đưa ra trong bảng dưới đây. Thêm vào đó bạn có thể xem và sửa đường dẫn của MATLAB (matlabpath). Những đường dẫn này chỉ cho MATLAB nơi chứa script file và hàm M_file trong máy tính của bạn. Có rất nhiều trường hợp các hàm trong MATLAB là các M_file đơn giản được chứa trong ổ đĩa, nhưng MATLAB thông báo không biết hàm này, như vậy do nó không tìm được đường dẫn của MATLAB, bạn cần phải thay đổi lại đường dẫn:

Các hàm hệ thống file	
addpath dir1	Thêm thư mục dir1 vào bắt đầu của đường dẫn
cd	Hiển thị thư mục hiện thời
p = cd	Gán thư mục làm việc hiện thời cho biến p
cd path	Thay đổi thư mục đưa ra bằng đường dẫn
delete test.m	Xoá M_file test.m
dir	Danh sách tất cả các file trong thư mục hiện thời
d = dir	Trả lại file trong thư mục hiện thời trong cấu trúc biến d
edit test	Mở test.m để soạn thảo, giống như Open trong bảng chọn file
exist('cow','file')	Kiểm tra sự tồn tại của file cow.m trong đường dẫn
exist('d','dir')	Kiểm tra sự tồn tại của thư mục d trong đường dẫn
filesep	Tách file như '\' trong Windows95 và NT, ':' trên Macintosh
fullfile	Tạo tên file với đường dẫn đầy đủ
inmem	Danh sách hàm M_file, gọi ra từ bộ nhớ
ls	Giống như dir

MATLABrc m	MATLAB chủ khởi động script M_file, thực hiện trước khi startup m
MATLABroot	Trả đường dẫn thư mục cho chương trình thực hiện MATLAB
path	Hiện thị hoặc sửa đường dẫn của MATLAB (MATLABpath)
pathdef m	Ham M_file, nơi mà matlabpath là dung
pathsep	Chia đường dẫn cho matlabpath
pwd	Giống như cd
rmpath dir1	Bỏ đi thư mục dir1 từ đường dẫn matlabpath
startup m	script M_file thực hiện khi MATLAB khởi động
tempdir	Tên của thư mục tạm thời
tempname	Tên của file tạm thời
type test	Hiện ra M_file test.m trong cửa sổ lệnh
what	Trả lại danh sách tất cả M_file và MAT_file trong thư mục hiện thời
which test	Hiện thị đường dẫn thư mục đến test.m

Đường dẫn của MATLAB là danh sách của tất cả các thư mục lưu trữ các file của MATLAB. Hơn nữa, nếu bạn tạo một thư mục của M_file thì đường dẫn của nó phải được thêm vào matlabpath, nếu không thì MATLAB không thể truy cập đến các file của bạn được, trừ khi file đó đặt trong thư mục hiện thời.

Để xem MATLAB sử dụng matlabpath như thế nào, hãy xem trường hợp được mô tả trong bảng sau:

Đường dẫn của MATLAB
<p>Khi bạn gõ >> cow, MATLAB sẽ làm như sau:</p> <ol style="list-style-type: none"> (1) Kiểm tra nếu cow là một biến trong không gian làm việc của MATLAB, nếu không thì... (2) Nó kiểm tra nếu cow là một ham được xây dựng, nếu không thì.. (3) Nó kiểm tra nếu một tên M_file cow.m tồn tại trong thư mục hiện thời, nếu không thì... (4) Nó kiểm tra nếu cow.m tồn tại bất cứ nơi nào trên đường dẫn của MATLAB bằng cách tìm kiếm đường dẫn.

Khi nào tìm thấy sự phù hợp thì MATLAB chấp nhận nó. Ví dụ như *cow* (ồn tại như một biến trong không gian làm việc của MATLAB, thì MATLAB không dùng hàm hoặc biến có tên là *cow*. Vì vậy bạn tránh không nên tạo biến có tên trùng với tên hàm như:

```
>> sqrt = 1.2;
```

```
>> sqrt(2);
```

Những lệnh trên sẽ tạo ra lỗi, bởi vì *sqrt* ở đây không phải là hàm tính căn bậc hai, nó là biến có giá trị là 1.2. Thủ tục đường dẫn còn được dùng khi dùng lệnh **load**. Đầu tiên MATLAB tìm kiếm trong thư mục hiện tại, sau đó nó tìm theo đường dẫn của MATLAB đến file dữ liệu.

Thực tế thủ tục tìm kiếm của MATLAB phức tạp hơn phần đã trình bày ở trên rất nhiều vì MATLAB dùng rất nhiều file có phần mở rộng là '.m'. Hàm *M_file* có thể chứa nhiều hơn một biến, thư mục trong *matlabpath* có thể có thư mục con gọi là **private**, và MATLAB cung cấp chương trình hướng đối tượng với các toán tử định nghĩa lại *M* file ở trong thư mục con, bắt đầu bằng ký tự @. Nếu tất cả những đặc điểm này được cộng thêm vào bảng trên thì nó sẽ đầy đủ hơn, nhưng sẽ rất khó hiểu. Nếu bạn muốn nghiên cứu thêm về phần này có thể xem các tài liệu cung cấp trong đĩa CD.

Nếu bạn có *M_file* hoặc *MAT_file* chứa trong thư mục không phải ở trong đường dẫn của MATLAB và không ở trong thư mục hiện tại, MATLAB không thể tìm thấy chúng. Có hai giải pháp cho vấn đề này là:

(1) Tạo thư mục nết kế thành thư mục hiện tại, dùng lệnh **cd** hoặc **pwd** từ trong bảng trước. —

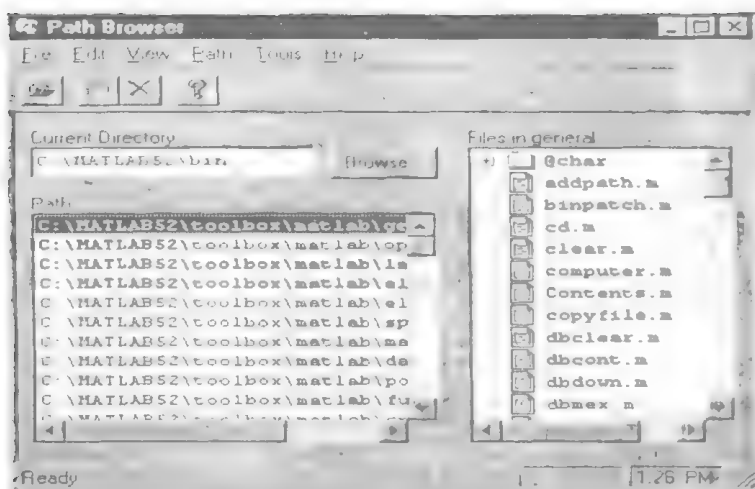
(2) Công thêm thư mục thiết kế trong đường dẫn của MATLAB.

Cuối cùng nó rất dễ dàng khi ta sử dụng phương pháp duyệt qua các đường dẫn (**path browser**) hoặc các lệnh trong cửa sổ lệnh **path** và **addpath**. Để dùng **path browser**, ta chọn **set path** từ bảng chọn file hoặc nhấn chuột trên nút **path browser** trên thanh công cụ của cửa sổ lệnh. Làm như vậy ta sẽ được màn hình giống như hình 5.1.

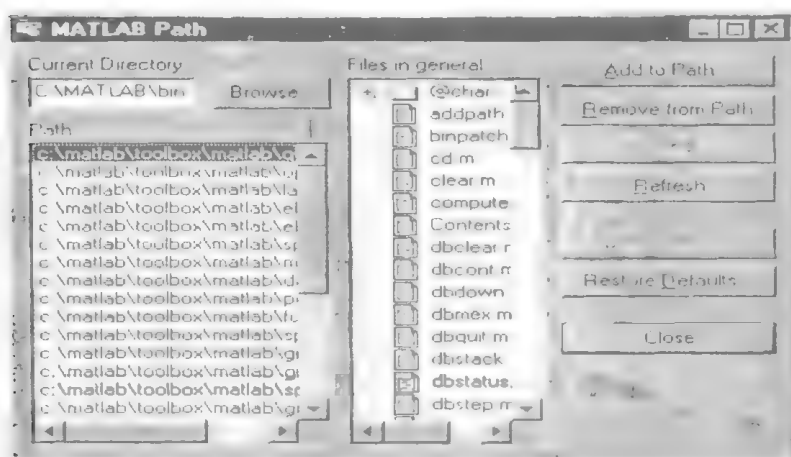
Giống như thiết kế các GUI, đường dẫn liên quan trực tiếp khi ta sử dụng. Đường dẫn *matlabpath* được hiển thị ở bên trái, thư mục con nằm trong đường dẫn được chọn nằm ở bên trái, còn các nút thay đổi đường dẫn như thêm đường dẫn mới (**add to path**), loại bỏ đường dẫn (**remove from path**) ở phía trên. Để ghi lại sự thay đổi ta chọn **save path** từ bảng chọn file của cửa sổ

path browser trước khi đóng GUI.

Cửa sổ **path browser** trong MATLAB 5.0 không khác lắm so với MATLAB 5.2, chủ yếu là các nút thay đổi đường dẫn trong MATLAB 5.2 thì nó đặt ở trên đỉnh còn ở MATLAB 5.0 nó được đặt ở bên phải. Để ghi lại sự thay đổi đường dẫn trong MATLAB 5.0 trước khi đóng GUI ta nhấn nút **save settings** (hình 5.2).



Hình 5.1 Path browser trong MATLAB 5.2



Hình 5.2 Path browser trong MATLAB to Student

MATLAB khi khởi động

Khi khởi động MATLAB, nó tạo ra hai script M_file là matlabrc.m và startup.m, trong đó matlabrc.m đi cùng MATLAB, và nhìn chung là không được sửa nó.

Các lệnh trong M_file tạo một cấu hình mặc định về kích cỡ của cửa sổ và vị trí của nó, cũng như các đặc điểm mặc định khác trong Windows95, WindowNT. Đường dẫn mặc định được tạo bằng cách gọi script file pathdef.m từ matlabrc.m. Trong các phần, các lệnh trong matlabrc.m kiểm tra sự tồn tại của script M_file startup.m trong đường dẫn của MATLAB nếu nó tồn tại, các lệnh trong nó được thực hiện.

Sự lựa chọn M_file startup.m chứa các lệnh có những đặc điểm riêng đối với MATLAB. Ví dụ nó rất thông thường nếu ta thêm một hoặc hơn các lệnh *path* hoặc *addpath* trong startup.m để chèn thêm các thư mục vào trong đường dẫn của MATLAB. Giống như vậy, mặc định hiển thị khuôn dạng số có thể thay đổi được như format compact. Nếu bạn có màn hình cân bằng xám, lệnh *graymon* sẽ có ích khi tạo mặc định đồ hoạ cho chế độ này. Hơn nữa, nếu bạn vẽ đồ thị có các kiểu mặc định riêng thì một sự gọi tới *colordef* có thể xuất hiện trong startup.m. Khi startup.m là một file chuẩn trong script M_file, thì không một lệnh nào có thể thay thế được trong nó. Tuy nhiên ta có thể thay thế lệnh quit trong startup.m.

CÁC PHÉP TOÁN ĐỐI VỚI MẢNG

Tất cả mọi sự tính toán đều có một điểm chung là có sử dụng đến các đại lượng vô hướng, gọi là scalars. Phép toán có liên quan đến scalars là các phép toán cơ bản, nhưng một lúc nào đó, phép toán phải lặp lại nhiều lần khi tính trên nhiều số. Để giải quyết vấn đề này, MATLAB định nghĩa thao tác trên mảng dữ liệu.

6.1 Mảng đơn

Giả sử ta xét hàm $y = \sin(x)$ trong một nửa chu kỳ ($\pi \geq x \geq 0$) trong khoảng này số điểm giá trị của x là vô tận, nhưng ta chỉ xét những điểm cách nhau một khoảng giá trị là 0.1π như vậy số các giá trị của x là đếm được. Từ đó ta có mảng các giá trị của x là:

$$x = 0, 0.1\pi, 0.2\pi, \dots, \pi$$

Nếu ta dùng máy tính kỹ thuật để tính thì ta được tương ứng các giá trị của y , từ đó ta có mảng của y :

x	0	0.1 π	0.2 π	0.3 π	0.4 π	0.5 π	0.6 π	0.7 π	0.8 π	0.9 π	π
y	0	0.31	0.59	0.81	0.95	1.0	0.95	0.81	0.59	0.31	0

trong mảng x chứa các phần tử x_1, x_2, \dots, x_{11}

trong mảng y chứa các phần tử y_1, y_2, \dots, y_{11}

Trong MATLAB để tạo những mảng này rất đơn giản; ví dụ để tạo hai mảng trên ta đánh các lệnh sau vào dấu nhắc của MATLAB:

```
>> x=[0 .1*pi .2*pi .3*pi .4*pi .5*pi .6*pi .7*pi .8*pi .9*pi pi]
```

```
x=
```

```
Columns 1 through 7
```

```
0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
```

Columns 8 through 11

2.1991 2.5133 2.8274 3.1416

```
>> y = sin(x)
```

y=

Columns 1 through 7

0 0.3090 0.5878 0.8090 0.9511 1.0000 0.9511

Columns 8 through 11

0.8090 0.5878 0.3090 0.0000

Kết quả trên ta được mảng của y gồm các phần tử tương ứng là sine của các phần tử của x, ở đây MATLAB ngầm hiểu là ta tính sine của từng phần tử của x.

Để tạo mảng, ta đặt các phần tử của mảng vào giữa hai dấu ngoặc vuông "[...]"; giữa hai phần tử của mảng có thể là dấu cách hoặc dấu phẩy ",".

6.2 Địa chỉ của mảng

Trên mảng x có 1 hàng, 11 cột hay có thể gọi là vector hàng, mảng có độ dài 11

+) Để truy nhập đến các phần tử của mảng ta dùng các chỉ số thứ tự của phần tử đó trong mảng

ví dụ x(1) là phần tử thứ nhất của mảng, x(2) là phần tử thứ hai của mảng...

```
>> x(2) % phần tử thứ nhất của mảng
```

ans=

0.3142

```
>> y(5) % phần tử thứ 5 của mảng
```

ans=

0.9511

+) Để truy nhập đến nhiều phần tử của mảng, ví dụ ta truy nhập từ phần tử thứ nhất đến phần tử thứ năm của mảng x:

```
>> x(1:5)
```

ans=

0 0.3142 0.6283 0.9425 1.2566

Truy nhập từ phần tử thứ / đến phần tử cuối của mảng y

```
>> y(7:end)
```

ans=

```
0.9511 0.8090 0.5878 0.3090 0.0000
```

Truy nhập từ phần tử thứ ba đến phần tử thứ nhất của mảng y:

```
>> y(3:-1:1)
```

ans=

```
0.5878 0.3090 0
```

Trong ví dụ trên 3 là phần tử thứ 3, 1 là chỉ phần tử đầu tiên, còn -1 là giá trị cộng (vị trí phần tử sau bằng vị trí phần tử trước cộng với -1)

Truy nhập đến các phần tử trong khoảng từ phần tử thứ 2, đến phần tử thứ 7, vị trí của phần tử sau bằng vị trí của phần tử trước cộng với 2, của mảng x:

```
>> x(2:2:7)
```

ans=

```
0.3142 0.9425 1.5708
```

Tạo mảng gồm các phần tử thứ 1, 2, 8, 9 của mảng y:

```
>> y([8 2 9 1])
```

ans=

```
0.8090 0.3090 0.5878 0
```

Nếu ta truy nhập vào các phần tử của mảng mà thứ tự các phần tử tăng đều với 1, ta có thể đánh lệnh:

```
>> x(1:3)
```

ans=

```
0 0.3142 0.6283
```

6.3 Cấu trúc của mảng

Với mảng có số lượng phần tử ít thì ta có thể nhập vào trực tiếp, nhưng với mảng có số lượng lớn các phần tử thì ta dùng một trong hai cách sau:

+) Tạo một mảng bắt đầu là phần tử 0, sau bằng phần tử trước cộng với 0.1, phần tử cuối là 1, tất cả các phần tử của mảng được nhân với π :

```
>> x = (0:0.1:1)*pi
```

```
x =
```

```
Columns 1 through 7
```

```
0 0.3142 0.6283 0.9425 1.2566 1.5708 1.8850
```

```
Columns 8 through 11
```

```
2.1991 2.5133 2.8274 3.1416
```

*) Tạo mảng gồm các phần tử của x bằng hàm *linspace*. Cú pháp của hàm này như sau:

linspace(giá trị phần tử đầu, giá trị phần tử cuối, số các phần tử)

Ví dụ

```
>> x = linspace(0,pi,11)
```

```
x =
```

```
Columns 1 through 7
```

```
0 0.3142 0.6283 0.9425 1.2566 1.5708 1.8850
```

```
Columns 8 through 11
```

```
2.1991 2.5133 2.8274 3.1416
```

Cách thứ nhất giúp ta tạo mảng mà chỉ cần vào khoảng cách giá trị giữa các phần tử (không cần biết số phần tử), còn cách thứ hai ta chỉ cần vào số phần tử của mảng (không cần biết khoảng cách giá trị giữa các phần tử).

Ngoài các mảng trên, MATLAB còn cung cấp mảng không gian theo logarithm bằng hàm *logspace*. Cú pháp của hàm *logspace* như sau:

logspace(số mũ đầu, số mũ cuối, số phần tử)

Ví dụ

```
>> logspace(0,2,11)
```

```
ans =
```

```
Columns 1 through 7
```

```
1.0000 1.5849 2.5119 3.9811 6.3096 10.0000 15.8489
```

```
Columns 8 through 11
```

```
25.1189 39.8107 63.0957 100.0000
```

Tạo mảng, giá trị bắt đầu tại 10^0 , giá trị cuối là 10^2 , chứa 11 giá trị.

Các mảng trên là các mảng mà các phần tử của nó được tạo nên theo một quy luật nhất định. Nhưng đôi khi mảng được yêu cầu mà không thể tạo các phần tử bằng các phương pháp trên, không có một mẫu chuẩn nào để tạo các mảng này. Trường hợp đó ta có thể tạo mảng bằng cách vào nhiều phần tử cùng một lúc.

Ví dụ

```
>> a = 1:5, b = 1:2:9
```

a=

```
1 2 3 4 5
```

b=

```
1 3 5 7 9
```

```
>> c = [a b]
```

```
1 2 3 4 5 1 3 5 7 9
```

Ở ví dụ trên ta đã tạo hai mảng thành phần là a và b sau đó tạo mảng c bằng cách ghép hai mảng a và b.

Ta cũng có thể tạo mảng như sau:

```
>> d=[a(1:2:5) 1 0 1]
```

d=

```
1 3 5 1 0 1
```

a là mảng gồm các phần tử [1 3 5], mảng d là mảng gồm các phần tử của a và ghép thêm các phần tử [1 0 1]

Tóm lại ta có bảng cấu trúc các mảng cơ bản:

$x = [2 \ 2\pi \ \sqrt{2} \ 2-3]$	Tạo vector hàng x chứa các phần tử đặc biệt.
$x = \text{first} : \text{last}$	Tạo vector hàng x bắt đầu tại first, phần tử sau bằng phần tử trước cộng với 1, kết thúc là phần tử có giá trị bằng hoặc nhỏ hơn last.
$x = \text{first} : \text{increment} : \text{last}$	Tạo vector hàng x bắt đầu tại first, giá trị cộng là increment, kết thúc là phần tử có giá trị bằng hoặc nhỏ hơn last.
$x = \text{linspace}(\text{first}, \text{last}, n)$	Tạo vector hàng x bắt đầu tại first, kết thúc là last, có n phần tử.
$x = \text{logspace}(\text{first}, \text{last}, n)$	Tạo vector hàng không gian logarithm x bắt đầu tại 10^{first} , kết thúc tại 10^{last} , có n phần tử.

6.4 Vector hàng và vector cột

Trong các ví dụ trước, mảng chứa một hàng và nhiều cột, người ta thường gọi là vector hàng. Ngoài ra ta còn có mảng là vector cột, tức là mảng có một cột và nhiều hàng, trong trường hợp này tất cả mọi thao tác và tính toán đối với mảng như ở trên là không thay đổi.

Từ các hàm tạo mảng minh họa ở phần trước (tất cả đều tạo vector hàng), có nhiều cách để tạo vector cột. Một cách trực tiếp để tạo vector cột là vào từng phần tử của mảng như ví dụ sau:

```
>> c = [1;2;3;4;5]
```

```
c=
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

Khác với trước là ta dùng dấu cách hay dấu phẩy để phân cách giữa hai cột của vector hàng. Còn ở ví dụ này ta dùng dấu chấm phẩy để phân cách giữa hai hàng của vector cột.

Một cách khác để tạo các vector cột là dùng các hàm *linspace*, *logspace*, hay từ các vector hàng, sau đó dùng phương pháp chuyển vị. MATLAB dùng toán tử chuyển vị là (') để chuyển từ vector hàng thành vector cột và ngược lại.

Ví dụ tạo một vector a và vector b là chuyển vị của vector a, vector c là chuyển vị của vector b:

```
>> a= 1:5
```

```
a=
```

```
1    2    3    4    5
```

```
>> b= a'
```

```
b=
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
>> c= b'
```

```
c=
```

```
1 2 3 4 5
```

Ngoài ra MATLAB còn sử dụng toán tử chuyển với dấu chấm đứng trước ('.') (toán tử chuyển vị chấm). Toán tử này chỉ khác với toán tử chuyển vị (') khi các phần tử của mảng là số phức tức là từ một vector nguồn với các phần tử là số phức, toán tử (') tạo ra vector phức liên hợp chuyển vị, còn toán tử ('.') chỉ tạo ra vector chuyển vị.

Ví dụ sau đây sẽ làm rõ điều trên:

```
>> c = a.' % Tạo vector c từ vector a ở trên bằng toán tử chuyển vị chấm
```

```
c=
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
>> d = a + i*a % Tạo vector số phức d từ vector a
```

```
d=
```

```
Columns 1 through 4
```

```
1.0000+1.0000i 2.0000+2.0000i 3.0000+3.0000i 4.0000+4.0000i
```

```
Columns 5
```

```
5.0000+5.0000i
```

```
>> e = d.' % Tạo vector e từ vector d bằng toán tử chuyển vị chấm ('.')
```

```
e=
```

```
1.0000 + 1.0000i
```

```
2.0000 + 2.0000i
```

```
3.0000 + 3.0000i
```

```
4.0000 + 4.0000i
```

```
5.0000 + 5.0000i
```

```
>> f = d' % Tạo ra vector f từ vector d bằng toán tử chuyển vị (')
```

```
f=
1.0000 - 1.0000i
2.0000 - 2.0000i
3.0000 - 3.0000i
4.0000 - 4.0000i
5.0000 - 5.0000i
```

Ở trên ta chỉ xét đến mảng có một hàng hay một cột bây giờ ta xét trường hợp có nhiều hàng và nhiều cột, nó còn được gọi là ma trận. Ví dụ sau đây là ma trận g có hai hàng và bốn cột:

```
>> g = [1 2 3 4;5 6 7 8]
```

```
g=
1 2 3 4
5 6 7 8
```

Trong ví dụ này ta dùng dấu cách để vào các phần tử trong hàng và dấu chấm phẩy (;) để tạo hai hàng; ngoài ra ta cũng có thể tạo ma trận như sau:

```
>> g = [1 2 3 4
5 6 7 8
9 10 11 12]
g=
1 2 3 4
5 6 7 8
9 10 11 12
```

Chú ý: Khi nhập vào ma trận thì giữa các hàng số phần tử phải bằng nhau nếu không chương trình sẽ bị báo lỗi như ví dụ sau:

```
>> h = [1 2 3;4 5 6 7]
```

Numbers of elements in each row must be the same

+) Phép toán giữa mảng với số đơn.

Trong ví dụ trước chúng ta đã tạo mảng x bằng cách nhân các phần tử của một mảng với π . Các phép toán đơn giản khác giữa mảng với số đơn là phép cộng, phép trừ, phép nhân, và phép chia của mảng cho số đó bằng cách thực hiện phép toán đối với từng phần tử của mảng.

Ví dụ:

```
>> g = [1 2 3 4; 5 6 7 8; 9 10 11 12];
```

```
>> -2 % Trừ các phần tử của mảng g đi 2
```

```
ans=
```

```
-1  0  1  2
 3  4  5  6
 7  8  9 10
```

```
>> 2*g - 1 % Nhân tất cả các phần tử của mảng g với 2 sau đó trừ đi 1
```

```
ans=
```

```
1  3  5  7
 9 11 13 15
17 19 21 23
```

+) Phép toán giữa mảng với mảng

Thuật toán thực hiện phép toán giữa các mảng không phải đơn giản như trên mà nó còn bị ràng buộc bởi các điều kiện khác như đối với hai mảng kích cỡ như nhau thì ta có các phép toán sau: phép cộng, phép trừ, phép nhân, chia tương ứng giữa các phần tử của của hai mảng.

Ví dụ

```
>> g % Gọi lại mảng g
```

```
g=
```

```
1  2  3  4
 5  6  7  8
 9 10 11 12
```

```
>> h = [1 1 1 1; 2 2 2 2; 3 3 3 3] % Tạo một mảng mới h.
```

```
h=
```

```
1  1  1  1
 2  2  2  2
 3  3  3  3
```

```
>> h + g % Cộng hai ma trận g và h (cộng tương ứng từng phần tử của h với g)
```

ans=

2	3	4	5
7	8	9	10
12	13	14	15

>> ans = h % Lấy kết quả trước trừ đi mảng h, ta được lại mảng g.

ans=

1	2	3	4
5	6	7	8
9	10	11	12

>> 2*g - h % Nhân ma trận g với 2 sau đó lấy kết quả trừ đi ma trận h.

ans=

1	3	5	7
8	10	12	14
15	17	19	21

>> g.*h % Nhân tương ứng các phần tử của mảng g với các phần tử của mảng h

ans=

1	2	3	4
10	12	14	16
27	30	33	36

Ở ví dụ trên ta đã dùng toán tử chấm_nhân (*), ngoài ra MATLAB còn dùng toán tử chấm_chia (/ hoặc \) để chia tương ứng các phần tử của hai mảng như ví dụ dưới đây:

>> g./h % Chia phải tương ứng các phần tử của mảng g với các phần tử của mảng h

ans=

1.0000	2.0000	3.0000	4.0000
2.5000	3.0000	3.5000	4.0000
3.0000	3.3333	3.6667	4.0000

>> h.\g % Chia trái tương ứng các phần tử của mảng g với các phần tử của mảng h

ans=

1.0000	2.0000	3.0000	4.0000
2.5000	3.0000	3.5000	4.0000
3.0000	3.3333	3.6667	4.0000

Chú ý ta chỉ có thể dùng phép nhân_chấm hay phép chia_chấm đối với các mảng g' và h mà không thể dùng phép nhân (*) hay phép chia (/ hoặc \) vì đối với các phép toán này yêu cầu số cột và số hàng của hai ma trận phải tương thích.

ví dụ:

```
>> g*h
```

```
??? Error using ==> *
```

```
Inner matrix dimensions must agree.
```

```
>> g/h
```

```
Warning: Rank deficient, rank = 1 tol = 503291e-15.
```

```
ans=
```

0	0	0.8333
0	0	2.1667
0	0	3.5000

```
>> h/g
```

```
Warning: Rank deficient, rank = 2 tol = 1.8757e-14.
```

```
ans=
```

-0.1250	0	0.1250
-0.2500	0	0.2500
-0.3750	0	0.3750

Phép chia ma trận đưa ra kết quả mà không cần thiết phải cùng kích cỡ như ma trận g và ma trận h. Về các phép toán đối với ma trận chúng ta sẽ nói đến sau

+) Mảng với lũy thừa.

MATLAB dùng toán tử (.^) để định nghĩa lũy thừa của mảng.

Ví dụ ta có hai mảng g và h như ở trên, ta có thể tạo các mảng mới bằng toán tử (.^) như sau:

```
>> g.^2 % Các phần tử của g được lũy thừa với số mũ là 2.
```

ans=

1 4 9 16
25 36 49 64
81 100 121 144

>> g.^-1 % Các phần tử của g được lũy thừa với số mũ là -1.

ans=

1 0.5 0.33333 0.25
0.2 0.16667 0.14286 0.125
0.11111 0.1 0.090909 0.083333

>> 2.^g % Các phần tử của g là số mũ của 2.

ans=

2 4 8 16
25 36 49 64
729 1000 1331 1728

>> g ^ (h - 1) % Các phần tử của g được lũy thừa với số mũ là tương ứng
các phần tử của h trừ đi 1.

ans=

1 1 1 1
5 6 7 8
81 100 121 144

Sau đây là bảng một số phép toán cơ bản của mảng:

Các phép toán đối với các phần tử của mảng	
Dữ liệu minh họa: $a = [a_1 \ a_2 \ \dots \ a_n]$, $b = [b_1 \ b_2 \ \dots \ b_n]$, c là số vô hướng	
Cộng với số đơn	$a+c = [a_1+c \ a_2+c \ \dots \ a_n+c]$
Nhân với số đơn	$a*c = [a_1*c \ a_2*c \ \dots \ a_n*c]$
Cộng mảng	$a+b = [a_1+b_1 \ a_2+b_2 \ \dots \ a_n+b_n]$
Nhân mảng	$a.*b = [a_1.*b_1 \ a_2.*b_2 \ \dots \ a_n.*b_n]$
Chia phải mảng	$a./b = [a_1./b_1 \ a_2./b_2 \ \dots \ a_n./b_n]$

Cnĩa trái mảng	$a \setminus b = [a_1 \setminus b; a_2 \setminus b; \dots a_n \setminus b_n]$
Lũy thừa mảng	$a.^c = [a_1.^c \ a_2.^c \ \dots \ a_n.^c]$ $c.^a = [c.^{a_1} \ c.^{a_2} \ \dots \ c.^{a_n}]$ $a.^b = [a_1.^{b_1} \ a_2.^{b_2} \ \dots \ a_n.^{b_n}]$

6.5 Mảng có các phần tử là 0 hoặc 1

MATLAB cung cấp những hàm để tạo những mảng mà các phần tử của chúng là 0 hoặc 1.

Ví dụ:

```
>> ones(3) % Tạo mảng 3 hàng, 3 cột với các phần tử là 1.
```

ans=

```
1 1 1
1 1 1
1 1 1
```

```
>> zeros(2,5) % Tạo mảng 2 hàng, 5 cột với các phần tử là 0.
```

ans=

```
0 0 0 0 0
0 0 0 0 0
```

Tạo mảng có các phần tử là 1, kích cỡ bằng mảng g đã biết.

```
>> size(g) % Hàm trả về kích cỡ của mảng g.
```

ans=

```
3 4
```

```
>> ones(size(g))
```

ans=

```
1 1 1 1
1 1 1 1
1 1 1 1
```

Khi gọi hàm **ones(n)**, **zeros(n)** với một thông số n thì MATLAB sẽ tạo mảng vuông với số hàng và số cột là n. Khi gọi hàm với hai thông số **ones(r,c)**, **zeros(r,c)** thì r là chỉ số hàng, c là chỉ số cột.

6.6 Thao tác đối với mảng

Từ các mảng và các ma trận cơ bản của MATLAB, có nhiều cách để thao tác đối với chúng. MATLAB cung cấp những cách tiện ích để chèn vào, lấy ra, sắp xếp lại những bộ phần tử con của chúng bằng các chỉ số của các phần tử. Ví dụ dưới đây sẽ minh họa những đặc điểm thao tác đối với mảng và ma trận ở trên.

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

```
A =
```

```
1  2  3
4  5  6
7  8  9
```

```
>> A(3,3) = 0    % Gán phần tử hàng thứ 3, cột thứ 3 bằng 0.
```

```
1  2  3
4  5  6
7  8  0
```

```
>> A(2,6) = 1    % Gán phần tử hàng thứ 2, cột thứ 6 bằng 1.
```

```
A =
```

```
1  2  3  0  0  0
4  5  6  0  0  1
7  8  0  0  0  0
```

Ở đây ma trận A không có 6 cột, kích cỡ của ma trận A phải tăng lên cho phù hợp, các phần tử lấp thêm được điền bằng các con số không.

```
>> A(:,4) = 4    % Gán tất cả các phần tử thuộc cột thứ 4 bằng 4.
```

```
A =
```

```
1  2  3  4  0  0
4  5  6  4  0  1
7  8  0  4  0  0
```

Ở trên ta dùng dấu hai chấm (:) để chỉ tất cả các hàng.

```
>> A = [1 2 3; 4 5 6; 7 8 9]; % Gán lại các giá trị của ma trận A.
```

```
>> B = A(3:-1:1,1:3) % Tạo ma trận B bằng cách đảo ngược các hàng của ma trận A.
```

B=

```
7  8  9
4  5  6
1  2  3
```

```
>> B = A(3:-1:1,:) % Cũng tạo ma trận B như trên
% nhưng ở đây ta dùng (:) để chỉ tất cả các cột.
```

B=

```
7  8  9
4  5  6
1  2  3
```

```
>> C = [ A B(:, [1 3])] % Tạo ma trận C bằng cách ghép ma trận A và
% cột thứ nhất, thứ ba của ma trận B vào bên phải ma trận A.
```

C=

```
1  2  3  7  9
4  5  6  4  6
7  8  9  1  3
```

```
>> C = [1 3]
```

C=

```
1 3
```

```
>> B = A(C,C) % Dùng ma trận C làm chỉ số để tạo ma trận B từ ma trận A.
```

B=

```
1  3
7  9
```

```
>> B = A(:) % Tạo ma trận cột B từ ma trận A.
```

B=

```
1
4
7
2
5
```

8
3
6
9

>> B = B.' % Chuyển ma trận B thành ma trận hàng bằng toán tử chuyển vị chấm.

B=

1 4 7 2 5 8 3 6 9

>> B = A;

>> B(:,2) = [] % Loại bỏ cột thứ hai của ma trận B.

B=

1 3

4 6

7 9

Khi ta gán cột thứ hai của ma trận B cho ma trận rỗng ([]) thì nó sẽ bị xóa, ma trận còn lại sẽ rút bỏ đi hàng thứ hai.

>> B = B.'

B=

1 4 7

3 6 9

>> B(2,:) = []

B=

1 4 7

>> A(2,:) = B % Thay hàng thứ hai của ma trận A bằng ma trận B.

A=

1 2 3

1 4 7

7 8 9

>> B = A([2 2 2])

B-

```
2  2  2  2
4  4  4  4
8  8  8  8
```

Tạo ma trận B bằng cách tạo bốn cột giống cột thứ hai của ma trận A, số hàng vẫn giữ nguyên bằng số hàng của ma trận A.

```
>> A(2,2) = []
```

??? Indexed empty matrix assignment is not allowed.

Ở đây MATLAB không cho phép xoá đi một phần tử của ma trận mà phải xoá đi một cột hoặc một hàng.

```
>> B = A(4,:)
```

??? Index exceeds matrix dimension.

Ví dụ trên ma trận A không có bốn hàng, nên MATLAB thông báo như trên.

```
>> B(1:2,:) = A
```

??? In an assignment A(matrix, :) = B, the number of columns in A and B must be the same.

MATLAB chỉ ra rằng bạn không thể gán một ma trận vào trong một ma trận khác mà khác nhau về kích cỡ.

```
>> B = [1 4 7];
```

```
>> B(3:4,:) = A(2:3,:)
```

B=

```
1  4  7
0  0  0
1  4  7
7  8  9
```

Nhưng ta có thể gán hai hàng của ma trận A cho hai hàng của ma trận B, khi ma trận A và ma trận B có cùng số cột. Ma trận B chỉ có một hàng nên khi thêm hàng thứ ba và hàng thứ tư thì hàng thứ hai của ma trận B được mặc định cho thêm các phần tử 0 vào.

```
>> G(1:6) = A(:,2:3)
```

G=

2 4 8 3 7 9

Từ phần tử thứ nhất đến phần tử thứ sáu của ma trận G được gán bằng cột thứ hai và cột thứ ba của ma trận A.

Đôi khi để tiện lợi hơn ta chỉ dùng chỉ số đơn để truy nhập đến các phần tử của mảng. Khi chỉ số đơn được dùng trong MATLAB thì thứ tự các phần tử của mảng được tính bắt đầu từ phần tử đầu tiên của cột, tính hết cột thì tính đến cột tiếp theo.

Ví dụ:

```
>> D = [1 2 3 4; 5 6 7 8; 9 10 11 12]
```

D=

1 2 3 4

5 6 7 8

9 10 11 12

```
>> D(2)        % Phần tử thứ hai của mảng.
```

ans=

5

```
>> D(5)        % Phần tử thứ năm của mảng (cột 2, hàng 2).
```

ans=

6

```
>> D(end)      % Phần tử cuối cùng của mảng.
```

ans=

12

```
>> D(4:7)      % Từ phần tử thứ tư đến phần tử thứ bảy của ma trận.
```

ans=

2 6 10 3

Ngoài trường hợp dùng địa chỉ dựa trên bảng chỉ số, chúng ta còn có thể dùng địa chỉ dựa trên mảng logic là kết quả từ các phép toán logic. Nếu kích cỡ của mảng logic cân bằng với mảng tạo ra nó thì đó chính là địa chỉ của mảng. Trong trường hợp này thì phần tử True (1) được giữ lại và phần tử False (0) bị bỏ đi.

Ví dụ:

```
>> x = -3:3    % Tạo mảng dữ liệu.
```

```
x=
```

```
    -3    -2    -1     0     1     2     3
```

```
>> abs(x)>1
```

```
ans=
```

```
     1     1     0     0     0     1     1
```

Trả về một mảng logic với giá trị một tại những phần tử có trị tuyệt đối lớn hơn một.

```
>> y = x(abs(x)>1)
```

```
y=
```

```
    -3    -2     2     3
```

Tạo mảng y bằng cách lấy những phần tử của x mà có trị tuyệt đối lớn hơn một.

```
>> y = x([1 1 0 0 0 1 1])
```

??? Index into matrix is negative or zero. See release notes on
changes to logical indices

Câu lệnh bị lỗi mặc dù `abs(x)>1` và `[1 1 0 0 0 1 1]` cũng là vector như nhau. Trong trường hợp này, `[1 1 0 0 0 1 1]` là một mảng số, không phải là mảng logic. Vì vậy MATLAB cố đánh địa chỉ các phần tử có số chỉ số trong mảng `[1 1 0 0 0 1 1]` và câu lệnh bị lỗi vì không có phần tử 0. Tuy nhiên MATLAB cung cấp hàm *logical* để chuyển đổi từ mảng số sang mảng logic

```
>> y = x(logical([1 1 0 0 0 1 1]))
```

```
y=
```

```
    -3    -2     2     3
```

mảng logic làm việc với ma trận cũng như là đối với vector:

```
>> B = [5 -3; 2 -4]
```

```
B=
```

```
     5     -3
```

```
     2     -4
```

```
>> x = abs(B)>2
```

x=

1 1

0 0

>> y = B(x)

5

-3

4

Tuy nhiên kết quả được chuyển thành vector cột vì không cách nào để định nghĩa ma trận chỉ có ba phần tử.

Địa chỉ của mảng

A(r, c) Địa chỉ một mảng con trong mảng A, định nghĩa bằng các chỉ số vector của hàng thiết kế trong r, chỉ số vector của cột thiết kế trong c.

A(r, :) Địa chỉ một mảng con trong mảng A, định nghĩa bằng các chỉ số vector của hàng thiết kế trong r, và tất cả các cột của A.

A(:, c) Địa chỉ một mảng con trong mảng A, định nghĩa bằng tất cả các hàng của A, chỉ số vector của cột được thiết kế trong c.

A(:) Địa chỉ tất cả các phần tử của A như một vector cột, bằng cách ghép thứ tự các cột của vector A.

A(i) Địa chỉ một mảng con trong mảng A, định nghĩa bằng các chỉ số vector đơn được thiết kế trong i, với giả sử A là vector cột.

A(x) Địa chỉ một mảng con trong mảng A, định nghĩa bởi mảng logic x. x phải cùng kích cỡ với A.

6.7 Tìm kiếm mảng con

Nhiều khi chúng ta muốn biết các chỉ số hay danh sách các chỉ số của những phần tử của một mảng mà nó thoả mãn một biểu thức quan hệ, trong MATLAB để thực hiện việc đó ta sử dụng hàm *find*. hàm này trả về danh sách con chỉ số tại những phần tử mà biểu thức quan hệ của chúng là đúng:

>> x = -3:3

x=

-3 -2 -1 0 1 2 3


```
>> k = find(abs(x)>1)
```

```
k=
```

```
1 2 6 7
```

tìm những chỉ số tại nhưng vị trí mà tại đó $\text{abs}(x)>1$

```
y = x(k)
```

```
y=
```

```
-3 -2 2 3
```

Tạo mảng y, dùng các chỉ số trong mảng k.

Hàm *find* cũng có thể sử dụng trong ma trận:

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

```
A=
```

```
1 2 3
```

```
4 5 6
```

```
7 8 9
```

```
>> [i,j] = find(A>5)
```

```
i=
```

```
3
```

```
3
```

```
2
```

```
3
```

```
j=
```

```
1
```

```
2
```

```
3
```

```
3
```

Ở đây i là chỉ số hàng, còn j là chỉ số cột; giữa i và j có mối quan hệ tương ứng để chỉ những vị trí mà tại đó biểu thức quan hệ là đúng.

Chú ý: khi MATLAB trả lại hai hoặc nhiều biến, chúng được đặt trong dấu ngoặc vuông, và được đặt bên trái dấu bằng. Cú pháp này khác với cú pháp thao tác đối với mảng ở trên. Khi mà

[,j] được đặt bên phải dấu bằng, và nó xây dựng lên một mảng mà j được kết nối vào bên phải dấu bằng.

Bảng tóm tắt dạng lệnh của phần tìm kiếm mảng:

Tìm kiếm mảng	
<code>i = find(x)</code>	Trả lại các chỉ số của mảng x nơi mà các phần tử của nó khác không
<code>[r, c] = find(x)</code>	Trả lại chỉ số hàng và chỉ số cột của mảng x nơi mà các phần tử của nó khác không.

6.8 So sánh mảng

Chúng ta có thể dùng hàm *isequal* so sánh hai mảng. Thí dụ:

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

```
A=
```

```
1    4    7
```

```
2    5    8
```

```
3    6    9
```

```
>> B = A.*(-1).^A
```

```
B=
```

```
-1    4   -7
```

```
2   -5    8
```

```
-3    6   -9
```

```
>> C = 1:9 % Tạo mảng có cùng giá trị với A nhưng có khuôn dạng khác.
```

```
1    2    3    4    5    6    7    8    9
```

```
>> isequal(A,C)
```

```
ans=
```

```
0
```

```
>> isequal(A,B)
```

```
ans=
```

```
0
```

```
>> isequal(A,A)
```

```
ans=
```

```
1
```

```
>> isequal(C,C')
```

```
ans=
```

```
0
```

Hàm *isequal* trả lại giá trị logic là đúng (1) khi hai mảng có cùng kích cỡ, các phần tử giống nhau. Ngoài ra nó trả lại giá trị là sai (0).

Thêm vào đó, hàm *ismember* chỉ ra các phần tử giống nhau giữa hai mảng:

```
>> ismember(A,B) % Kết quả trả về là vector cột.
```

```
ans=
```

```
0
```

```
1
```

```
0
```

```
1
```

```
0
```

```
1
```

```
0
```

```
1
```

```
0
```

```
>> ismember(A,B)
```

```
ans=
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

1
1
1

ismember trả lại giá trị đúng cho những chỉ số ở trong A mà phần tử này cũng có ở trong đối số thứ hai. Hai đối số không cần có cùng kích cỡ.

```
>> x = 0:2:20 % mảng với 11 phần tử.
```

```
x=
```

0 2 4 6 8 10 12 14 16 18 20

```
>> ismember(x,A)
```

```
ans=
```

0 1 1 1 1 0 0 0 0 0 0

đây là mảng có cùng kích cỡ với x, với 1 tại các phần tử chung.

```
>> ismember(x,A)
```

```
ans=
```

0
1
0
1
0
1
0
1
0

Đây là mảng có số phần tử bằng số phần tử của A, với 1 tại các phần tử chung. Vì vậy *ismember* so sánh đối số thứ nhất của nó với đối số thứ hai và trả lại một vector có cùng số phần tử với đối số thứ nhất.

Những hàm tạo khác trong thư viện MATLAB:

```
>> union(A,B) % Tất cả các phần tử có trong hai mảng.
```

```
ans=
```

```
-9
```

```
-7
```

```
-5
```

```
-3
```

```
-1
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
>> intersect(A,B)    % Phần tử chung của hai mảng.
```

```
ans=
```

```
2
```

```
4
```

```
6
```

```
8
```

```
>> setdiff(A,B)    % Các phần tử có trong A nhưng không có trong B.
```

```
ans=
```

```
1
```

```
3
```

```
5
```

```
7
```

```
9
```

```
>> setxor(A B) % Các phần tử không thuộc phân chung giữa A và B
```

```
ans=
```

```
-9
```

```
-7
```

```
-5
```

```
-3
```

```
-1
```

```
1
```

```
3
```

```
5
```

```
7
```

```
9
```

Những hàm này được tổng kết lại trong bảng dưới đây:

So sánh mảng	
isequal(A, B)	Đúng nếu A và B giống nhau.
ismember(A, B)	Đúng khi phần tử của A cũng là phần tử của B.
intersect(A, B)	Các phần tử chung giữa A và B
setdiff(A, B)	Các phần tử có trong A mà không có trong B.
setxor(A, B)	Các phần tử không thuộc phân chung giữa A và B.
union(A, B)	Tất cả các phần tử có trong A và B.

6.9 Kích cỡ của mảng

Phân trước chúng ta đã biết lệnh **who** cung cấp tên biến do người dùng định nghĩa. Trong trường hợp của mảng, nó còn rất quan trọng khi biết kích cỡ của mảng. Trong MATLAB, lệnh **whos** cung cấp những thông tin này:

```
>> whos
```

Name	size	Bytes	Class
A	3x3	72	double array
B	1x3	24	double array
ans	1x4	32	double array (logical)

Grand total is 16 elements using 128 bytes

Thêm vào đó để đánh số và kích cỡ của biến, *whos* hiển thị tổng số bytes đã chiếm, và class của các biến. Ví dụ, ở thông tin để cập trên ans là mảng logic.

Trong những trường hợp mà kích cỡ của ma trận hoặc, của vector không được biết nhưng nó cần thiết cho một số các thao tác, MATLAB cung cấp hai hàm ứng dụng là *size* và *length*:

```
>> A = [1 2 3 4; 5 6 7 8];
```

```
>> s = size(A)
```

```
s =
```

```
2 4
```

Với một thông số ra, hàm *size* trả lại một vector hàng trong đó có hai phần tử, phần tử thứ nhất là chỉ số hàng, còn phần tử thứ hai chỉ số cột.

```
>> [r,c] = size(A)
```

```
r =
```

```
2
```

```
c =
```

```
4
```

Với hai thông số đưa ra, hàm *size* trả lại số hàng ở trong biến thứ nhất, và số cột ở trong biến thứ hai.

```
>> r = size(A,1)
```

```
r =
```

```
2
```

```
>> c = size(A,2)
```

Gọi hai thông số, hàm *size* chỉ trả về số cột hoặc số hàng.

```
>> length(A)
```

```
ans =
```

```
4
```

Trả về giá trị số hàng hoặc số cột, giá trị nào lớn hơn được trả về.

```
>> B = pi:0.01:2*pi;
```

```
>> size(B)
```

ans=

1 315

Cho biết rằng B là vector hàng, và

>> length(B)

ans=

315

trả lại độ dài của vector.

>> size([])

chỉ ra rằng ma trận rỗng không có kích cỡ.

Những khái niệm này được tổng kết trong bảng dưới đây:

Kích cỡ của mảng	
whos	Hiển thị các biến, ma tồn tại trong không gian làm việc và kích cỡ của chúng.
s = size(A)	Trả lại vector hàng s, mà phân tử thứ nhất là số hàng của A, phân tử thứ hai là số cột của A.
[r, c] = size(A)	Trả lại hai số vô hướng r, c chứa số hàng và số cột của A.
r = size(A, 1)	Trả lại số hàng của A trong biến r.
c = size(A, 2)	Trả lại số cột của A trong biến c.
n = length(A)	Trả lại max(size(A)) trong biến n khi A không rỗng

6.10 Mảng nhiều chiều

Đối với các MATLAB versions trước 5.0, mảng chỉ có thể có một hoặc hai chiều. Từ MATLAB 5.0 trở lên thì số chiều của mảng đã tăng lên. Ví dụ.

>> a = [1 0; 0 1]

a=

1 0

0 1

>> b = [2 2; 2 2]

b=

2 2

2 2


```

>> c = [0 3; 3 0]

c=

     0     3
     3     0

>> d = cat(3,a,b,c)

d( ...1)=

     1     0
     0     1

d(:, :,2)=

     2     2
     2     2

d(:, :,3)=

     0     3
     3     0

>> size(d)

ans=

     2     2     3

```

Tạo các mảng hai chiều a, b, c, sau đó ghép chúng lại với nhau thành mảng ba chiều bằng cách sử dụng hàm **cat**. Như vậy mảng d là mảng có hai hàng, hai cột, và ba trang. Mảng a tạo trang thứ nhất, b là trang thứ hai, và c là trang thứ ba. Thông số trang diễn tả chiều thứ ba của mảng, cung cấp một cách hình dung về mảng ba chiều như mảng hai chiều, các trang xếp thứ tự từ một cho đến cuối như trong một quyển sách. Đối với các mảng có số chiều cao hơn, không có tên chung, và nó cũng rất khó tưởng tượng!

Thao tác với mảng nhiều chiều cũng giống như các thủ tục đưa ra ở trên đối với mảng một chiều và hai chiều. Ngoài ra MATLAB còn cung cấp một số hàm thao tác trực tiếp đối với mảng nhiều chiều:

Các hàm với mảng nhiều chiều	
<code>s = size(A)</code>	Cho n_số chiều của A, trả về vector hàng s với n phần tử, phần tử thứ i là kích cỡ chiều thứ i của mảng A
<code>ndims(A)</code>	Số chiều của A, tương tự như hàm <code>length(size(A))</code>
<code>permute(A, order)</code>	n_số chiều, tương đương với toán tử chuyển vị chấm.

ipermute(A, order)	Ngược với hàm permute(A, order)
shiftdim(A, n)	Thay đổi số chiều của mảng A bằng số nguyên n.
squeeze(A)	Trả lại số chiều duy nhất của mảng, tương đương với trả lại số chiều lớn hơn ba.

Ví dụ: Sự suy giảm do phân rã dùng mảng

Vấn đề: Phân tử polonium có chu kỳ phân rã là 140 ngày, có nghĩa là do sự phân rã mà khối lượng của polonium chỉ còn lại 1/ 2 so với khối lượng ban đầu sau 140 ngày. Giả sử ban đầu ta có 10 gam polonium, nó sẽ còn lại bao nhiêu sau mỗi tuần trong vòng mười tuần?

Giải pháp: Ta sử dụng phương pháp giải trong chương 2, khối lượng còn lại sau một khoảng thời gian là:

$$\text{khối lượng còn lại} = \text{khối lượng ban đầu} \cdot (0.5)^{\text{thời gian/ chu kỳ}}$$

Để giải bài toán này, giải pháp của MATLAB là.

```
>> initial_amount = 10; % Khối lượng chất polonium ban đầu
```

```
>> half_life = 140; % Chu kỳ phân rã
```

```
>> time = 7 7 70 % Kết thúc của các tuần
```

```
time=
```

```
7 14 21 28 35 42 49 56 63 70
```

```
>> amount_left = initial_amount*0.5.^(time/ half_life)
```

```
amount_left=
```

```
Columns 1 through 7
```

```
9.6594 9.3303 9.0125 8.7055 8.4090 8.1225 7.8458
```

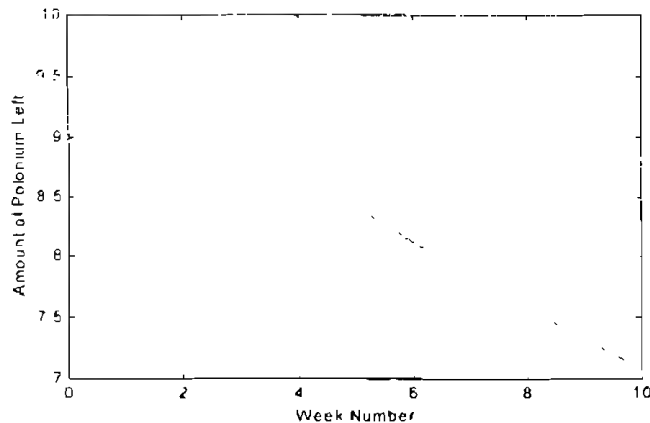
```
Columns 8 through 10
```

```
7.5786 7.3204 7.0711
```

Dùng toán tử mảng làm cho nó tính các giá trị một cách đơn giản hơn khi nhân nhiều giá trị của một biến. Chú ý rằng nhân chấm (.) được sử dụng vì chúng ta muốn lũy thừa 0.5 lên đối với mỗi phần tử của mảng. Những dữ liệu này được biểu diễn trong MATLAB như hình 6.1:

```
>> plot(time/7,amount_left)
```

```
>> xlabel('Week number') ylabel('Amount of Polonium left')
```



Hình 6.1 Số Polonium còn lại sau từng tuần

Ví dụ: Tìm kiếm giải pháp sử dụng vectors

Vấn đề: “Vấn đề của tuần” trong trường cấp hai là tìm: một số nhỏ hơn 100 mà chia hết cho 7, nhưng còn dư lại 1 khi chia cho 2, 3, 4, 5, và 6.

Giải pháp: Không có một giải pháp phân tích nào cho vấn đề này cả, vì vậy chúng ta phải giải bằng phương pháp tìm kiếm. Nếu bạn bắt đầu với tất cả các số là bội số của 7 và nhỏ hơn 1000, còn các số khác thì không xét đến, bạn sẽ xây dựng được một giải pháp. Trong MATLAB giải pháp được đưa ra trong script file là:

```
function pow
% pow.m script file to solve problem of the week
n=7:7:1000 % all multiples of 7 less than 1000
number=length(n) % number of potential solutions
n(rem(n,2)~=1)=[]; % throw out non solutions by
number=length(n)
n(rem(n,3)~=1)=[]; %setting them equal to an empty array,
number=length(n)
n(rem(n,4)~=1)=[]; % the function rem computes remainders
number=length(n)
n(rem(n,5)~=1)=[];
number=length(n)
```

```
n(mod(n,6)~=1)=[];
```

Chạy script file này ta được giả pháp như ở dưới đây:

```
>> pow
```

```
number =
```

```
142
```

```
number =
```

```
71
```

```
number =
```

```
24
```

```
number =
```

```
12
```

```
number =
```

```
2
```

```
n=
```

```
301 721
```

Ví dụ: Tính toán nồng độ axit dùng các phép toán với mảng

Vấn đề: Như một phần của quá trình sản xuất chi tiết của vật đúc tại một nhà máy tự động, chi tiết đó được nhúng trong nước để làm nguội, sau đó nhúng trong bồn đựng dung dịch axit để làm sạch. Trong toàn bộ của quá trình nồng độ axit giảm đi khi các bộ phận được lấy ra khỏi bồn axit vì khi nhúng chi tiết của vật đúc vào bồn thì một lượng nước còn bám trên vật đúc khi nhúng ở bể trước cũng vào theo và khi nhấc ra khỏi bồn một lượng axit bám theo vật. Để đảm bảo chất lượng thì nồng độ axit phải không được nhỏ hơn một lượng tối thiểu. Bạn hãy bắt đầu với nồng độ dung dịch là 90% thì nồng độ tối thiểu phải là 50%. Lượng chất lỏng thêm vào và lấy đi sau mỗi lần nhúng dao động trong khoảng từ 1% đến 10%. Hỏi bao nhiêu chi tiết có thể nhúng vào bể nước axit trước khi nồng độ của nó giảm xuống dưới mức cho phép?

Giải pháp: Ta sử dụng phương pháp giải đưa ra ở chương 2:

$$n = \frac{\log(\text{initial_con}/\text{min_con})}{\log(1 + \text{lost})}$$

Trong MATLAB, giải pháp viết trong script M_file là:

```
function example6_2
```

```
% script M file example6_2
```

```

initial_con=90;
min_con=50;
lost=1:10 % consider 1% to 10% in increments of 1%
n=floor(log(initial_con/min_con)./log(1+lost/100))
stem(lost,n)
xlabel('Percent Lost with Each Dip')
ylabel('Number of Dips')
title('Acid-Water Bath Dipping Example')

```

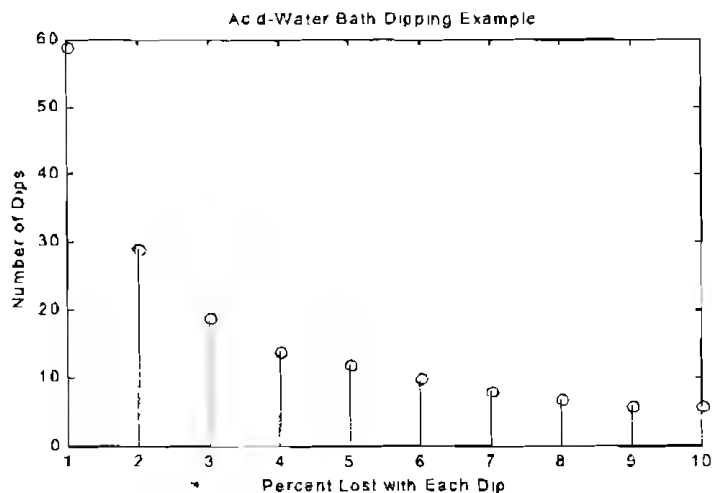
Chạy chương trình trên ta được kết quả như sau:

lost =

1 2 3 4 5 6 7 8 9 10

n =

59 29 19 14 12 10 8 7 6 6



Hình 6.2 Mức axit cho phép

Chú ý ở đây yêu cầu phương pháp chia chấm vì $\log(1 + \text{lost}/100)$ là một vector.

CÁC THAO TÁC VỚI MẢNG

7.1 Tạo phương trình tuyến tính

Một vấn đề thường gặp của số học tuyến tính là việc giải phương trình. Ví dụ giải phương trình sau dưới dạng ma trận:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 366 \\ 804 \\ 351 \end{bmatrix}$$

$$A \cdot x = b$$

Biểu tượng phép nhân toán học (.) được định nghĩa trong phép toán trên, khác với kí hiệu ta dùng đối với mảng trước kia. Trong MATLAB phép nhân ma trận này được định nghĩa bằng dấu sao (*). Tiếp theo định nghĩa dấu bằng, ma trận tạo ra từ ma trận A và vector x bằng với vector b. Thêm nữa, khi lời giải không tồn tại, có rất nhiều cách gần đúng để tìm kiếm giải pháp, như phép loại trừ Gaussian, sự tìm thừa số LU, hoặc tính trực tiếp $A^{-1} \cdot b$. Dưới đây chúng ta sẽ đề cập đến một số cách giải quyết như trên

Trước tiên nhập vào ma trận A và b:

```
>> A = [1 2 3; 4 5 6; 7 8 0]
```

```
A=
```

```
1    2    3
```

```
4    5    6
```

```
7    8    0
```

```
>> b = [366; 804; 315]
```

```
b=
```

366

804

351

Có thể kiểm tra xem định thức của ma trận trên có khác không hay không:

```
>> det(A)
```

ans=

27

Nếu nó đúng, MATLAB có thể giải phương trình theo hai cách, một cách hay được dùng hơn một cách ít sử dụng, nhưng trực tiếp hơn, phương pháp này là chuyển thành dạng $x = A^{-1}b$.

```
>> x = inv(A)*b
```

x=

25.0000

22.0000

99.0000

Ở đây *inv(A)* là hàm của MATLAB dùng để tính A^{-1} ; và toán tử nhân (*), không có dấu chấm phía trước, đây là phép nhân ma trận. Phương pháp được dùng nhiều hơn là dùng toán tử chia ma trận trái:

```
>> x = A\b
```

x=

25.0000

22.0000

99.0000

Phương trình này sử dụng phương pháp tìm thừa số LU gần đúng và đưa ra câu trả lời như là phép chia trái A cho b . Toán tử chia trái (\) không có dấu chấm phía trước là một phép toán của ma trận, nó không phải là các phép toán giữa các phần tử của mảng. Phương pháp thứ hai này được sử dụng nhiều hơn do nhiều nguyên nhân, một trong những nguyên nhân đầu tiên là phương pháp này dùng ít phép toán hơn và tốc độ nhanh hơn. Thêm vào đó, nhìn chung phương pháp này chính xác hơn cho những bài toán lớn. Trong trường hợp khác, nếu MATLAB không tìm thấy phương pháp giải hoặc không tìm thấy phương pháp chính xác, nó sẽ hiển thị thông báo lỗi.

Nếu bạn nghiên cứu số học tuyến tính, bạn biết rằng khi số phương trình và số biến khác nhau, thì không thể có một phương pháp duy nhất để giải. Trong MATLAB khi gặp những hệ phương trình có số phương trình lớn hơn số biến nó dùng toán tử chia trái hoặc chia phải, tự động giảm thấp nhất nhưng phân tử thừa $A \cdot x = b$. Cách này gọi là phương pháp vuông nhỏ nhất. Ví dụ.

```
>> A = [1 2 3; 4 5 6; 7 8 0; 2 5 8] % Bốn phương trình, ba biến.
```

```
A=
```

```
1    2    3
4    5    6
7    8    0
2    5    8
```

```
>> b = [366 804 351 514]'
```

```
b=
```

```
366
804
351
514
```

```
>> x = A\b % Phương pháp vuông nhỏ nhất.
```

```
x=
```

```
247.9818
-173.1091
114.9273
```

```
>> res = A*x - b
```

```
res=
```

```
-119.4545
11.9455
0.0000
35.8364
```


Mặt khác khi số phương trình ít hơn số biến tương tự như trường hợp không xác định, thì số nghiệm phương trình là vô tận. Đối với những nghiệm này MATLAB tính theo hai cách. Dung toán tử chia đưa ra phương pháp mà có số phần tử 0 của x là cực đại. Như một sự lựa chọn, tính $x = \text{pinv}(A) * b$ đưa ra phương pháp chiều dài hoặc tiêu chuẩn của x nhỏ hơn các phương pháp khác. Phương pháp này gọi là phương pháp tiêu chuẩn cực tiểu.

Ví dụ:

```
>> A = A' % Tạo ba phương trình, bốn biến.
```

```
A=
```

```
1   4   7   2
2   5   8   5
3   6   0   8
```

```
>> b = b(1:3)
```

```
b=
```

```
366
804
351
```

```
>> x = A\b % phương pháp với số phần tử 0 cực đại.
```

```
x=
```

```
0
-165.9000
99.0000
168.3000
```

```
>> xn = pinv(A)*b % Tìm kiếm giải pháp tiêu chuẩn nhỏ nhất.
```

```
xn=
```

```
30.8182
-168.9818
99.0000
159.0545
```

```
>> norm(x) % Tiêu chuẩn O_2 với các phần tử 0.
```

ans=

256.2200

>> norm(xn) % Giải pháp tiêu chuẩn nhỏ nhất

ans=

254.1731

7.2 Các hàm ma trận

Để giải phương trình tuyến tính, MATLAB cung cấp các hàm trợ giúp sau:

Các hàm ma trận	
balance(A)	Cân bằng để tăng độ chính xác
cdf2rdf(A)	Chuyển từ dạng số phức chéo sang dạng số thực chéo
chol(A)	Tìm thừa số Cholesky
cholinc(A, droptol)	Thừa số Cholesky không đầy đủ
cond(A)	Số điều kiện ma trận
condest(A)	Ước lượng số điều kiện ma trận theo tiêu chuẩn 1
det(A)	Định thức ma trận
expm(A)	Ma trận theo luật mũ
expm1(A)	Bổ sung M_file của expm
expm2(A)	Ma trận theo luật hàm mũ, dùng thứ tự Taylor
funm(A, 'fun')	Tính toán hàm ma trận chung
hess(A)	Mẫu Hessenberg
inv(A)	Ma trận chuyển vị
logm(A)	Ma trận logarithm
lu(A)	Tìm thừa số với phép khử Gaussian
luinc(A, droptol)	Thừa số LU không đầy đủ
norm(A)	Ma trận và vector tiêu chuẩn
norm(A,1)	Tiêu chuẩn 1

<code>norm(A, 2)</code>	Tiêu chuẩn 2
<code>norm(A, inf)</code>	Vô cùng
<code>norm(A, p)</code>	Tiêu chuẩn P (chỉ đối với vector)
<code>norm(A, 'fro')</code>	Tiêu chuẩn F
<code>normest(A)</code>	Tiêu chuẩn 2 ước lượng cho ma trận lớn
<code>null(A)</code>	Khoảng rỗng
<code>orth(A)</code>	Tính trực giao
<code>poly(A)</code>	Đa thức đặc trưng
<code>polyvalm(A)</code>	Tính giá trị của ma trận
<code>qr(A)</code>	Xác định trực giao tam giác
<code>qrdelete(Q, R, j)</code>	Xoá cột từ thừa số QR
<code>qrinsert(Q, R, j, x)</code>	Chèn cột trong thừa số QR
<code>rank(A)</code>	Số của hàng hoặc cột độc lập
<code>rcond(A)</code>	Ước lượng điều kiện thuận nghịch
<code>sqrtm(A)</code>	Ma trận gốc bình phương
<code>subspace(A, B)</code>	Góc giữa hai điểm
<code>svd(A)</code>	Phân tích giá trị đơn
<code>svds(A, K)</code>	Một số các giá trị đơn
<code>trace(A)</code>	Tổng các phần tử chéo

7.3 Ma trận đặc biệt

MATLAB đưa ra một số các ma trận đặc biệt. Dưới đây là một số trong các ma trận đó. Một số trong chúng có những ứng dụng rộng rãi trong các phép toán.

```
>> a = [1 2 3; 4 5 6];
>> b = find(a>10)
b=
[]
```

Ở đây b là ma trận rỗng. MATLAB trả lại ma trận rỗng khi phép toán không có kết quả. Trong ví dụ trên không có phần tử nào của a lớn hơn 10. Ma trận rỗng không có kích cỡ, nhưng tên biến của chúng vẫn tồn tại trong không gian làm việc.

```
>> zeros(3) % Ma trận không 3 hàng, 3 cột (3x3).
```

```
ans=
```

```
0 0 0
0 0 0
0 0 0
```

```
>> ones(2,4) % Ma trận một 2 hàng, 4 cột (2x4).
```

```
ans=
```

```
1 1 1 1
1 1 1 1
```

```
>> zeros(3) + pi
```

```
ans=
```

```
3.1416 3.1416 3.1416
3.1416 3.1416 3.1416
3.1416 3.1416 3.1416
```

Ví dụ trên về tạo ma trận 3x3 với các phần tử đều là π .

```
>> rand(3,1)
```

```
ans=
```

```
0.2190
0.0470
0.6789
```

ma trận 3x1 gồm các phần tử là số cung cấp bởi hàm random giữa 0 và 1.

```
>> randn(2)
```

```
ans=
```

```
1.1650 0.0751
0.6268 0.3516
```

Ma trận 2x2 của các số cung cấp bởi hàm random với giá trị trung bình là 0. Thuật toán cho hàm **rand** và **randn** có thể tìm thấy trong S.K. Park and K.W. Miller "Random Number Generator. Good Ones Are Hard to Find," Comm. ACM, 32, 10, Oct. 1988-1201

```
>> eye(3)
```

```
ans=
```

```
1  0  0
0  1  0
0  0  1
```

Ma trận đồng nhất 3x3

```
>> eye(3,2)
```

```
ans=
```

```
1  0
0  1
0  0
```

Ma trận đồng nhất 3x2

Ngoài ra để chỉ kích cỡ của một ma trận, bạn có thể dùng hàm **size** để tạo một ma trận có kích cỡ giống như ma trận khác:

```
>> A = [1 2 3; 4 5 6];
```

```
>> ones(size(A))
```

```
ans=
```

```
1  1  1
1  1  1
```

ma trận một có cùng kích cỡ với ma trận A.

Các ma trận trên và các ma trận đặc biệt khác được giới thiệu trong bảng sau:

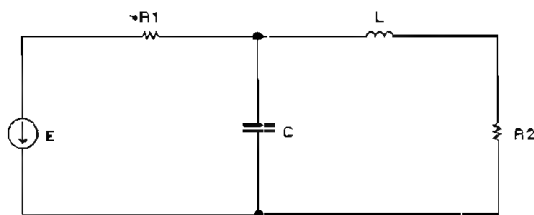
Các ma trận đặc biệt	
[]	Ma trận rỗng
companion	Tạo ma trận rỗng
eye	Ma trận đồng nhất
gallery	Ma trận kiểm tra nhỏ vài phần tử
hadamard	Ma trận Hadamard

hankel	Ma trận Hankel
hilb	Ma trận Hilbert
invhilb	Chuyển thành ma trận Hilbert
magic	Ma trận vuông, giá trị các phần tử bằng từ 1 đến giá trị số phần tử
ones	Ma trận 1
pascal	Ma trận tam giác Pascal
rand	Ma trận với các phần tử ngẫu nhiên từ 0 đến 1.
randn	Ma trận ngẫu nhiên thông thường với giá trị trung bình bằng 0
rosser	Ma trận kiểm tra đối xứng trực chính
toeplitz	Ma trận Toeplitz
vander	Ma trận Vandermonde
wilkinson	Ma trận kiểm tra Wilkinson
zeros	Ma trận không

Ví dụ

Vấn đề. Ta có mạch điện như trong hình 7.1 được mô tả bằng phương trình điện áp nút khi nguồn đưa vào là sóng hình sin.

$$\begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & 0 \\ -\frac{1}{2} & \left(\frac{1}{2} + j0.2 + \frac{1}{j10}\right) & -\frac{1}{j10} \\ 0 & -\frac{1}{j10} & \frac{1}{10} + \frac{1}{j10} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$



Hình 7.1 Sơ đồ mạch điện.

$$E = 1\angle 0^\circ; \quad R1 = 2; \quad L = 10j; \quad C = \frac{1}{0.2j}; \quad R2 = 10.$$

Ở đây v_i là điện áp giữa nút thứ i và đất. Hỏi điện áp tại mỗi nút ra bao nhiêu?

Giải pháp: Đây là vấn đề về phân tích pha. Phương pháp giải bài này là giải phương trình trên, và chuyển các kết quả về dạng thời gian. Trong MATLAB giải pháp sẽ là:

function circuit

% circuit.m script file to solve circuit problem

A(1,1)=1/2; % poke in nonzero values as needed

A(1,2)=-1/2;

A(2,1)=-1/2;

A(2,2)=1/2 + 0.2j + 1/10j;

A(2,3)= -1/10j;

A(3,2)=-1/10j;

A(3,3)=1/10 + 1/10j;

y=[-1 0 0]'; % right hand side vector

v=A\y % complex solution

vmag=abs(v) % solution magnitudes

vphase=angle(v)*180/pi % solution phase in degrees

theta=linspace(0,2*pi); % plot results in time

v1=vmag(1)*cos(theta-vphase(1));

v2=vmag(2)*cos(theta-vphase(2));

v3=vmag(3)*cos(theta-vphase(3));

thd=theta*180/pi; %

plot(thd,v1,thd,v2,thd,v3)

Sau khi chạy chương trình trên, kết quả sẽ là:

v =

-4.0000 + 6.0000i

-2.0000 + 6.0000i

2.0000 + 4.0000i

vmag =

7.2111

6.3246

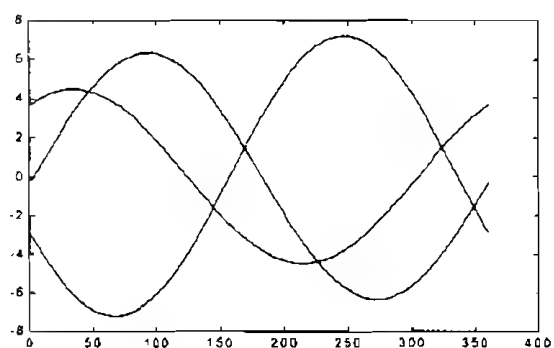
4.4721

vphase =

123.6901

108.4349

63.4349



Hình 7.2 Kết quả được biểu thị theo dạng sóng hình sin.

CÁC PHÉP TÍNH LOGIC VÀ QUAN HỆ

Ngoài những toán tử 'truyền thống', MATLAB cung cấp toán tử logic và quan hệ. Bạn có thể quen thuộc với những phép toán này nếu bạn đã làm quen với các ngôn ngữ lập trình khác. Mục đích của những toán tử và hàm này là để trả lời câu hỏi True/False (đúng/sai).

Đối với các số thì trong toán tử logic và quan hệ quy định các số khác không là True còn số không là False. Kết quả của phép toán logic và quan hệ đưa ra là 1 cho True, 0 cho False.

8.1 Toán tử quan hệ

Toán tử quan hệ MATLAB bao gồm tất cả các phép so sánh:

Toán tử quan hệ	Ý nghĩa
<	nhỏ hơn
<=	nhỏ hơn hoặc bằng
>	lớn hơn
>=	lớn hơn hoặc bằng
==	bằng
~=	gần bằng

Toán tử quan hệ MATLAB có thể dùng để so sánh hai mảng có cùng kích cỡ hoặc so sánh một mảng với một số đơn. Trong trường hợp thứ hai, số đơn so sánh với tất cả các phần tử của mảng, kết quả trả về giống như kích cỡ của mảng. Ví dụ:

```
>> A = 1:9; B = 9 - A
```

A=

1 2 3 4 5 6 7 8 9

B=

8 7 6 5 4 3 2 1 0

>> tf = A>4

tf=

0 0 0 0 1 1 1 1 1

tim kiểm các phần tử của A mà lớn hơn 4. Kết quả bằng 0 khi $A \leq 4$, bằng 1 khi $A > 4$.

>> tf = (A==B)

tf=

0 0 0 0 0 0 0 0 0

Tim kiểm các phần tử của A mà bằng với B. Chú ý sự khác nhau giữa $=$ và $==$ dùng để so sánh hai biến và trả về 1 khi chúng bằng nhau, 0 khi chúng khác nhau; $=$ dùng để gán kết quả đưa ra của toán tử cho một biến.

>> tf = B - (A>2)

tf=

8 7 5 4 3 2 1 0 -1

Tim các phần tử $A > 2$ và bị trừ bởi vector B. Ví dụ này chỉ ra rằng kết quả đưa ra của toán tử logic là một mảng số bao gồm các số không và một, chúng cũng có thể dùng trong các phép toán số học.

>> B = B + (B==0)*eps

B=

Columns 1 through 7

8.0000 7.0000 6.0000 5.0000 4.0000 3.0000 2.0000

Columns 8 through 9

1.0000 0.0000

Ví dụ trên đưa ra cách thay thế các phần tử của B mà trùng với không bằng số đặc biệt của MATLAB là eps, có giá trị xấp xỉ $2.2e-16$. Cách thay thế này đôi khi có ích là tránh trường hợp chia cho số không như ví dụ sau:

```
>> x = (-3:3)/3
      -1.0000  -0.6667  -0.3333   0   0.3333   0.6667   1.0000
>> sin(x)./x
Warning: Divide by zero
ans=
      0.8415   0.9276   0.9816   NaN  0.9816   0.9276   0.8415
```

Tính toán hàm $\sin(x)/x$ đưa ra một cảnh báo vì phân tử thứ tư bằng không, $\sin(0)/0$ không được định nghĩa, MATLAB trả lại NaN (nghĩa là không phải là một số) tại vị trí đó trong kết quả. Thử lại ví dụ trên, sau khi thay thế phân tử có giá trị bằng không bằng số eps.

```
>> x = x + (x==0)*eps;
>> sin(x)/x
ans=
      0.8415   0.9276   0.9816   1.0000   0.9816   0.9276   0.8415
```

Bây giờ $\sin(x)/x$ tại $x = 0$ đưa ra kết quả giới hạn chính xác.

8.2 Toán tử Logic

Toán tử logic cung cấp một cách diễn đạt mối quan hệ phủ định hay tổ hợp. Toán tử logic MATLAB bao gồm:

Toán tử logic	Ý nghĩa
&	AND
	OR
~	NOT

Một vài ví dụ về dùng toán tử logic:

```
>> A = 1:9; B = 9 - A;
>> tf = A>4
tf=
      0   0   0   0   1   1   1   1   1
```

Tìm kiếm các phần tử của A mà lớn hơn 4.

```
>> tf = ~(A>4)
```

```
1 1 1 0 0 0 0 0
```

phủ định của kết quả, tương đương với vị trí nào bằng không thay bằng một và ngược lại

```
>> tf = (A>2)&(A<6)
```

```
tf=
```

```
0 0 1 1 1 0 0 0
```

Trả lại một tại những vị trí mà phần tử của A lớn hơn 2 và nhỏ hơn 6.

8.3 Các hàm logic và hàm quan hệ

Thêm vào những toán tử logic và toán tử quan hệ để cấp đến ở trên, MATLAB cung cấp các hàm logic và quan hệ khác dưới đây:

Các hàm logic và hàm quan hệ khác	
xor(x,y)	Toán tử hoặc. Trả lại giá trị 1 khi x hoặc y khác không (True), giá trị 0 khi cả x và y cùng bằng không (False) hoặc cùng khác không (True)
any(x)	Trả lại 1 nếu bất cứ phân tử nào trong vector x khác không. Trả lại 1 cho mỗi cột trong ma trận x mà có các phần tử khác không.
all(x)	Trả lại 1 nếu tất cả các phân tử của vector x khác không. Trả lại 1 cho mỗi cột trong ma trận x mà tất cả các phân tử khác không.

MATLAB còn cung cấp rất nhiều các hàm kiểm tra cho sự tồn tại của các giá trị đặc biệt hoặc điều kiện và trả lại những kết quả là giá trị logic.

Các hàm kiểm tra	
isa(X, name)	True nếu X có lớp đối tượng là 'name'
iscell(X)	True nếu đối số là mảng phần tử.
iscellstr(X)	True nếu đối số là mảng phần tử của các chuỗi.
ischar(S)	True nếu đối số là chuỗi kí tự.
isempty(X)	True nếu đối số là rỗng.
isequal(A, B)	True nếu A và B giống nhau.
isfield(S, 'name')	True nếu 'name' là một trường của cấu trúc S

isfinite(X)	True khi các phần tử có hạn.
isglobal(X)	True khi đối số là biến toàn cục
ishandle(h)	True khi đối số là sự điều khiển đối tượng hợp lý.
ishold	True nếu đồ thị hiện tại giữ trạng thái ON.
isiee	True nếu máy tính thực hiện phép số học IEEE
isinf(X)	True tại những phần tử vô cùng
isletter(S)	True khi các phần tử thuộc bảng chữ cái.
islogical(X)	True khi đối số là mảng logic
ismember(A, B)	True tại những vị trí mà phần tử của A và B trùng nhau
isnan(X)	True khi các phần tử là không xác định (NaN)
isnumeric(X)	True khi đối số là mảng số
isppc	True cho Macintosh với bộ xử lý PowerPC
isprime(X)	True khi các phần tử là số nguyên tố
isreal(X)	True khi đối số không có phần ảo
isspace(S)	True khi các phần tử là kí tự trắng
issparse(A)	True nếu đối số là ma trận Sparse
isstruct(S)	True nếu đối số là một cấu trúc
isstudent	True nếu Student Edition của MATLAB
isunix	True nếu máy tính là UNIX
isvms	True nếu máy tính là VMS

VĂN BẢN

Sự tiện ích của MATLAB là xử lý với các con số. Tuy nhiên chúng ta đã nhiều lần đề cập đến thao tác với văn bản (text) như khi đưa nhãn và tiêu đề vào trong đồ thị. Trong MATLAB biến text được dùng đến như là xâu kí tự, hoặc đơn giản là các xâu.

9.1 Xâu kí tự

Xâu kí tự trong MATLAB là mảng của các giá trị ASCII mà quy ước của nó là các kí tự.

Ví dụ:

```
>> t = 'How about this character string?'
t=
How about this character string?
>> size(t)
ans=
     1     32
>> whos
Name      Size      Bytes   Class
t         1x32       64      char array
Grand total is 32 elements using 64 bytes
```

Một xâu kí tự, đơn giản là dạng văn bản, được đặt giữa hai dấu nháy đơn. Mỗi kí tự trong xâu là một phần tử của mảng, với mỗi phần tử chiếm hai bytes.

Muốn xem các mã ASCII của một xâu kí tự, bạn phải dùng các phép toán số học đối với xâu, hoặc chuyển nó sang dạng số, dùng hàm *double*. Ví dụ:

```
>> double(t)
ans=
```

Columns 1 through 12

```
72 111 119 32 97 98 111 117 116 32 116 104
```

Columns 12 through 24

```
105 115 32 99 104 97 114 97 99 116 101 114
```

Columns 25 through 32

```
32 115 116 114 105 110 103 63
```

```
>> abs(t)
```

```
ans=
```

Columns 1 through 12

```
72 111 119 32 97 98 111 117 116 32 116 104
```

Columns 13 through 24

```
105 115 32 99 104 97 114 97 99 116 101 114
```

Columns 25 through 32

```
32 115 116 114 105 110 103 63
```

Hàm *char* chuyển lại thành xâu:

```
>> char(t)
```

```
ans=
```

How about this character string?

Với mảng xâu là một mảng số với thuộc tính đặc biệt, chúng ta có thể thao tác bằng tất cả các công cụ thao tác với mảng sẵn có trong MATLAB. Ví dụ:

```
>> u = t(16:24)
```

```
u=
```

```
character      *
```

Địa chỉ của xâu cũng giống như mảng. Ở đây phần tử từ 16 đến 24 chứa từ character

```
>> u = t(24:-1:16)
```

```
retcarahc
```

Đây là từ "character" đọc ngược lại

```
>> u = t(16:24)'
```

```
u=
```

```
c
```

```
h
```

```
a
```

```
r
```

```
a
```

```
c
```

```
t
```

```
e
```

```
r
```

Dùng toán tử chuyển vị để chuyển từ “character” sang dạng ma trận cột

```
>> v = 'I can't find the manual!'
```

```
v=
```

```
I can't find the manual!
```

Dấu nháy đơn với xâu kí tự là biểu tượng trong hai dấu nháy đơn.

Chúng ta có thể nối hai xâu như đối với hai mảng:

```
>> w = {u,v}
```

```
w=
```

```
character I can't find the manual!
```

Hàm *disp* cho phép bạn hiển thị xâu kí tự mà không có tên biến

```
>> disp(v)
```

```
I can't find the manual
```

Chú ý là trạng thái “v=” bị bỏ đi, điều này rất có ích cho chúng ta hiển thị những lời trợ giúp trong script file.

Cũng giống như đối với ma trận, xâu kí tự có thể có nhiều hàng, nhưng mỗi một hàng phải có số cột bằng nhau, để cho số cột của chúng bằng nhau chúng ta có thể dùng kí tự trống.

```
>> v = ['However, this'
```

```
'does work!  ']
```



```
v=
```

```
However, this
```

```
does work!
```

```
>> w = ['this'; ' does not']
```

```
??? All rows in the bracketed expression must have the same  
number of columns.
```

```
>> size(v)
```

```
ans=
```

```
2    13
```

Ta cũng có thể dùng hàm *char* để tạo một mảng xâu từ các xâu, và nó tự thêm các kí tự trống để tạo ra một mảng đầy đủ.

```
>> w = char('this', 'does not')
```

```
w=
```

```
this
```

```
does not
```

```
>> size(w)
```

```
ans=
```

```
2     8
```

9.2 Chuyển đổi xâu

Để bổ xung thêm về sự chuyển đổi giữa xâu và mã ASCII của nó như đã trình bày ở trên, MATLAB đưa ra một số các hàm chuyển đổi hữu ích khác, chúng bao gồm dưới đây:

Các hàm chuyển đổi xâu	
base2dec	Dựa trên xâu x chuyển sang hệ mười.
bin2dec	Từ xâu nhị phân sang hệ mười
char	Từ xâu sang ASCII
dec2base	Từ hệ mười sang xâu x
dec2bin	Từ số hệ mười sang xâu nhị phân

dec2hex	Từ số hệ mười sang xâu của các số hệ mười sáu.
int2str	Chuyển từ mã ASCII sang xâu
fprintf	Viết dạng văn bản ra file hoặc ra màn hình
hex2dec	Chuyển từ xâu gồm các số hệ 16 sang các số hệ mười
hex2num	Chuyển từ xâu các số hệ 16 sang số dấu phẩy động IEEE
int2hex	Chuyển từ số nguyên sang xâu
mat2str	Chuyển từ ma trận số sang xâu gồm các số
num2str	Chuyển từ số sang xâu
sprintf	Chuyển từ mã ASCII sang xâu
sscanf	Chuyển từ số sang xâu có điều chỉnh kích thước
str2num	Chuyển từ xâu sang số không có điều chỉnh kích thước

Trong trường hợp chung ta tạo một thông báo có chứa các số không phải là xâu, những hàm chuyển đổi sẽ giúp chúng ta làm việc đó.

```
>> rad = 2.5; area = pi*rad^2;
>> t = ['A circle of radius ' num2str(rad)...
'has an area of ' num2str(area) '.'];
>> disp(t)
A circle of radius 2.5 has an area of 19.63.
```

Ở đây hàm **num2str** được dùng để chuyển từ số sang xâu. Giống như vậy **int2str** chuyển từ số nguyên sang xâu, cả hai hàm này gọi hàm **sprintf**, nó giống như cú pháp trong C dùng để chuyển số sang xâu.

9.3 Các hàm về xâu

MATLAB đưa ra một số các hàm của xâu, bao gồm các hàm trong danh sách dưới đây

Các hàm xâu	
blanks(n)	Trả lại một xâu gồm các kí tự trống hay dấu cách
deblank(s)	Trả lại các vệt trống từ một xâu

eval(xâu)	Ước lượng xâu như là một lệnh của MATLAB
eval(try, catch)	Ước lượng xâu và bắt lỗi
feval(f, x, y, ...)	Hàm evalute đưa ra bằng xâu
findstr(s1, s2)	Tìm kiếm một xâu trong một xâu khác
ischar(s)	True nếu đưa vào là một xâu
isletter(s)	True tại những vị trí kí tự Alphabet tồn tại
isspace(s)	True tại những vị trí là kí tự trống
lasterr	Xâu của lỗi cuối cùng MATLAB đưa ra
lower(s)	Xâu với những chữ cái thường
strcat(s1, s2, ...)	Nối các xâu thành hàng
strcmp(s1, s2)	True nếu các xâu giống nhau
strmatch(s1, s2)	Tìm kiếm khả năng giống nhau của xâu
strncmp(s1, s2, n)	True nếu n kí tự đầu giống nhau
strrep(s1, s2)	Thay thế một xâu bằng một xâu khác
strtok(s)	Tìm kiếm dấu hiệu cho xâu
strvcat(s1, s2, ...)	Nối các xâu thành cột
upper(s)	Chuyển thành chữ in

Một số các hàm trên cung cấp khả năng xử lý các xâu cơ bản. Ví dụ như, *findstr* trả lại chỉ số bắt đầu của một xâu trong một xâu khác

```
>> b = 'Peter Piper picked a peck of pickled peppers';
```

```
>> findstr(b, ' ') % Tìm kiếm khoảng trống
```

```
6 12 19 21 26 29 37
```

```
>> findstr(b, 'p')
```

```
9 13 22 30 38 40 41
```

```
>> find(b=='p')
```

```
9 13 22 30 38 40 41
```

```
>> findstr(b, 'cow') % Tìm kiếm từ cow
```

```
ans=
```

```
[]
```

```
>> findstr(b,'pick')
```

```
ans=
```

```
13 30
```

Hàm này trả lại ma trận rỗng khi không có những phần cần tìm.

```
>> strrep(b,'Peter','Pamela')
```

```
ans=
```

```
Pamela Piper picked a peck of pickled peppers
```

Như trình bày ở trên, strrep đơn giản chỉ là sự thay thế một xâu. strrep không làm việc với ma trận xâu, vì vậy trước tiên bạn cần phải chuyển từ ma trận thành vector.

9.4 Ma trận tế bào của xâu

Ma trận tế bào là một kiểu dữ liệu cho phép bạn gọi tên và thao tác với một nhóm dữ liệu có nhiều kích cỡ và nhiều kiểu.

```
>> C = {'How';'about';'this for a';'cell array of strings?'}
```

```
C=
```

```
'How'
```

```
'about'
```

```
'this for a'
```

```
'cell array of strings?'
```

```
>> size(C)
```

```
4 1
```

Ma trận trên có 4 hàng và một cột nhưng mỗi cột lại có độ dài khác nhau. Tất cả các phần tử được đặt trong dấu ngoặc nhọn, mỗi phần tử được đặt trong dấu nháy đơn, giữa hai hàng là dấu chấm phẩy. Mảng tế bào được đánh địa chỉ cũng giống như mảng thông thường:

```
>> C(2:)
```

```
ans=
```

```
'about'
```

```
'this for a'
```

```
>> C([4 3 2 1])
ans=
    'cell array of strings?'
    'this for a'
    'about'
    'How'
```

```
>> C(1)
```

```
ans=
    How
```

Đây vẫn là mảng tế bào. Để thay đổi dấu nhảy của tế bào, ta sử dụng ngoặc nhọn:

```
>> s = c{4}
ans=
    cell array of strings?
```

```
>> size(s)
```

```
ans=
     1     22
```

Để truy nhập vào nhiều hơn một tế bào, ta dùng hàm *deal*:

```
>> [a, b, c, d] = deal(C{:})
```

```
a=
```

```
How
```

```
b=
```

```
about
```

```
c=
```

```
this for a
```

```
d=
```

```
cell array of trings?
```

Ở đây C{:} để chỉ truy nhập đến tất cả các tế bào, nó giống như:

```
>> [a, b, c, d] = deal(C{1}, C{2}, C{3}, C{4})
```

```
a=
```

How

b=

about

c=

this for a

d=

cell array of strings?

Hàm *char* có thể dùng để chuyển từ mảng tế bào sang mảng xâu:

```
>> s = char(C)
```

How

about

this for a

cell array of strings?

```
>> size(s)    % Kết quả là các xâu với các khoảng trống.
```

ans=

```
    4    22
```

```
>> ss = char(C(1:2))
```

ss=

How

about

```
>> size(ss)
```

ans=

```
    2    5
```

Để chuyển ngược lại mảng tế bào, ta dùng hàm *cellstr*.

```
>> cellstr(s)
```

ans=

'How'

'about'

'this for a'

'cell array of strings?'

Hầu hết các hàm xâu trong MATLAB làm việc với cả mảng xâu hoặc mảng tế bào.

Về mảng tế bào sẽ được trình bày rõ hơn ở Chương 19.

THỜI GIAN

MATLAB đưa ra một số hàm thao tác về thời gian, từ đó bạn có thể tính toán với ngày, giờ, in lịch và tìm kiếm những ngày cụ thể. MATLAB chứa ngày và thời gian như một số có độ chính xác hai số sau dấu phẩy tượng trưng cho số ngày, bắt đầu bằng năm không. Ví dụ, mùng 1 tháng 1 năm 1997 tại lúc nửa đêm, nó được tượng trưng bởi số 729391, và cùng một ngày nhưng lúc buổi trưa là 729391.5. Cấu trúc này có thể dễ dàng cho máy tính xử lý, nhưng nó rất khó diễn giải. Do vậy MATLAB cung cấp các hàm trợ giúp chuyển đổi giữa số và xâu kí tự và để thao tác với ngày và thời gian.

10.1 Ngày và giờ hiện tại

Hàm *clock* trả về ngày và giờ hiện tại chứa trong một mảng. Ví dụ:

```
>> T = clock

T =

    1997     1    21    16    33   39.934708
```

Hàm *now* trả về ngày và thời gian hiện tại như số ngày quy ước của máy hoặc đơn giản là số ngày.

```
>> t = now

t =

    729411.690045541
```

Cả hai kết quả ở trên có cùng một thông tin.

Hàm *date* trả lại ngày hiện tại như một xâu theo mẫu: dd-mmm-yyyy

```
>> date

ans =

    21-Jan-1997
```


10.2 Sự chuyển đổi giữa các kiểu

Bạn có thể chuyển số ngày ra xâu, sử dụng hàm *datestr*. Cấu trúc của hàm này có dạng như sau:

```
datestr(date_number,format_spec).
```

Sau đây là trợ giúp của *help* cho hàm *datestr*.

```
>> help datestr
```

DATESTR string representation of date.

DATESTR(D,DATEFORM) converts a serial data number D (as returned by DATENUM) into a date string. The string is formatted according to the format number or string DATEFORM (see table below). By default, DATEFORM is 1, 16, or 0 depending on whether D contains

'dates, times or both.

DATEFORM number	DATEFORM string	Example
0	'dd-mmm-yyyy HH:MM:SS'	01-Mar-1995 15:45:17
1	'dd-mmm-yyyy'	01-Mar-1995
2	'mm/dd/yy'	03/01/95
3	'mmm'	Mar
4	'm'	M
5	'mm'	3
6	'mm/dd'	03/01
7	'dd'	1
8	'ddd'	Wed
9	'd'	W
10	'yyyy'	1995

11	'yy'	95
12	'mmyy'	Mar95
13	'HH MM:SS'	15:45:17
14	'HH:MM:SS PM'	3:45:17 PM
	'HH:MM'	15:45
16	'HH.MM PM'	3:45 PM
17	'QQ-YY'	Q1-96
18	'QQ'	Q1

Ví dụ với hàm *datestr*:

```
>> datestr(t)
ans=
    21-Jan-1997 16:33:40
>> datestr(t,14)
ans=
    4:33:40 PM
```

Hàm *datenum* là hàm ngược của *datestr*. Hàm này chuyển một chuỗi kí tự dạng ngày dùng mẫu *datenum(str)*, hoặc một số độc lập hoặc một vector sang số dạng ngày, dùng mẫu *datenum(year, month, day)* hoặc *datenum(year, month, day, hour, minute, second)*.

```
>> datenum('21-Jan-1997 16:33:40')
ans=
    729411.690045541
>> datenum(1997, 01, 21)
ans=
    729411
>> datenum(1997, 01, 21, 16, 33, 40)
ans=
    729411.690045541
```

Hàm *datevec* chuyển một chuỗi kí tự dạng ngày (dùng *datestr* dạng 0, 1, 2, 6, 13, 14, 15, hoặc 16) hoặc một số dạng ngày sang vector.

```

>> c = datevec('12/ 24/ 1984')
c=
    1984    12    24    0    0    0
>> [yr, mo, day, hr, nim, sec] = datevec('24-Dec-1984 08: 22')
yr=
    1984
mo=
    12
day=
    24
hr=
     8
min=
    22
sec=
     0

```

10.3 Các hàm về ngày

Ngày của tuần có thể tìm từ xâu dạng ngày hoặc số dạng ngày, dùng hàm *weekday*, MATLAB sử dụng quy ước Sunday = 1 và Saturday = 7.

```

>> [d w] = weekday(728647)
d=
     2
w=
    Mon
>> [d w] = weekday('21-Dec-1994')
d=
     4
w=
    Wed

```

Ngày cuối tháng có thể tìm bằng hàm *eomday*. Trong đó bắt buộc phải đưa vào năm, tháng.

```
>> eomday(1996, 2)    % 1996 là năm
```

```
ans =
```

```
29
```

MATLAB có thể tạo lịch cho bất cứ tháng nào bạn yêu cầu, và hiển thị nó trong cửa sổ lệnh hoặc đặt chúng trong một ma trận 6x7.

```
>> calendar('7/ 17/ 95')
```

Jul 1995

S	M	Tu	W	Th	F	S
0	0	0	0	0	0	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	0	0	0	0	0

```
>> S = calendar(1994, 12)
```

```
S =
```

0	0	0	0	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
0	0	0	0	0	0	0

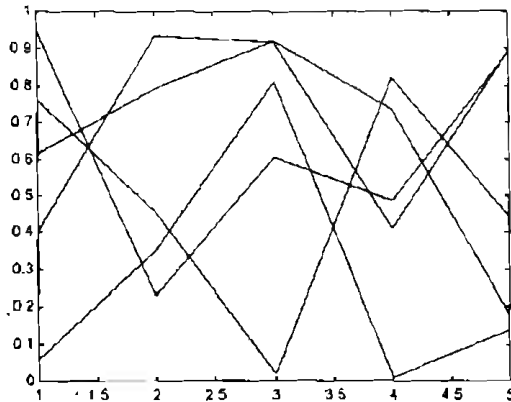
10.4 Các hàm về thời gian

Lệnh *tic* và *toc* có thể được dùng đối với thời gian trong tính toán:

```
>> tic; plot(rand(5)); toc
```

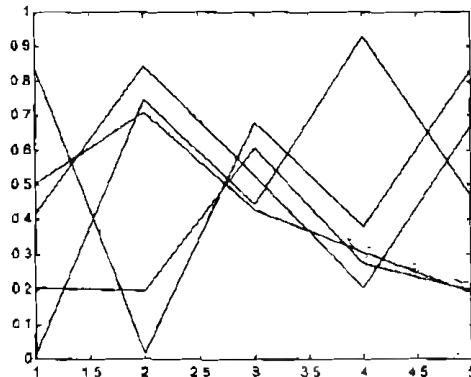
```
elapsed_time =
```

```
0.2200
```



Hình 10.1

```
>> tic; plot(rand(5)); toc
elapsed_time =
    0.1700
```



Hình 10.2

Chú ý sự khác nhau về hàm thời gian giữa `elapsed_time` đối với lệnh `plot`, lệnh `plot` thứ hai nhanh hơn vì MATLAB đã tạo hình dáng cửa sổ và dịch các hàm cần thiết vào trong bộ nhớ.

Hàm `cputime` trả về tổng số thời gian của CPU (Central Processing Unit), tính theo giây, trong thời gian MATLAB đã dùng từ khi nó được khởi động lên.

Hàm `etime` tính khoảng thời gian giữa hai vector thời gian. Các vector phải là vector hàng gồm 6 phần tử, giống như kết quả trả về trong lệnh `clock` và `datevec`. Tại thời gian hiện tại `etime` không chuyển giữa tháng và năm.

Tất cả các hàm có thể sử dụng để tính toán thời gian.

```
>> t0 = cputime; pause(5); cputime - t0
```

```
ans =
```

```
5
```

```
>> t1 = clock; pause(2); etime(clock,t1)
```

```
ans =
```

```
2.0400
```

Bạn hãy xem help và MATLAB CD để tìm hiểu thêm về những hàm này.

10.5 Vẽ đồ thị với hàm ngày và thời gian

Đôi khi nó rất có ích để vẽ đồ thị trong đó dùng trục ngày và thời gian cho một hoặc hơn một các nhân. Hàm **datetick** tự động với công việc này. Nếu đồ thị được vẽ, dùng số ngày cho một hoặc hơn một trục, thì hàm **datetick** sẽ viết các nhãn cho điểm đánh dấu. Ví dụ sau vẽ hình 10.3:

```
>> t = (1900:10:1990)';
```

```
>> p = [75.995; 91.972; 105.771; 123.203; 131.669;  
150.697; 179.323; 203.212; 226.505; 249.633];
```

```
>> plot(datenum(t,1,1),p)
```

```
>> datetick('x','yyyy') % use 4-digit year on the x-axis
```

```
>> title('Population by year')
```

Chúng ta có thể tạo biểu đồ cột của công ty bán hàng bán từ tháng 11 năm 1994 đến tháng 12 năm 1995 (Hình 10.4):

```
>> y = [1994 1994 1995*ones(1,12)]';
```

```
>> m = [11 12 (1:12)]';
```

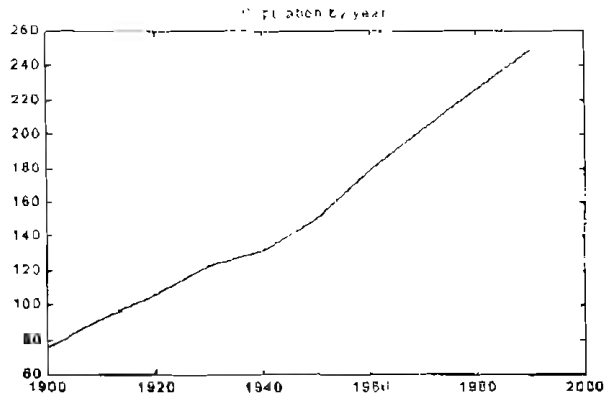
```
>> s=[1.1 1.3 1.2 1.4 .16 1.5 1.7 1.6 1.8 1.3 1.9 1.7 1.6 1.95]';
```

```
>> bar(datenum(y,m,1),s)
```

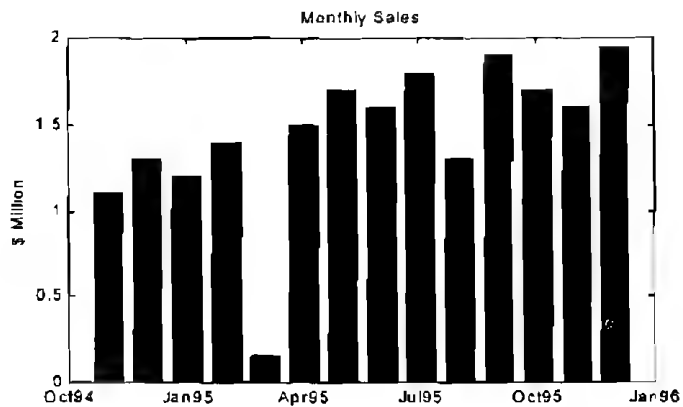
```
>> datetick('x','mmyy')
```

```
>> ylabel('$ Million')
```

```
>> title('Monthly Sales')
```



Hình 10.3



Hình 10.4

Ví dụ: Tìm thứ sáu ngày 13

Bây giờ chúng ta đã được giới thiệu các lệnh về thời gian, hãy dùng chúng để tạo một số hàm có ích. Nếu bạn là người cẩn thận, bạn muốn biết bao giờ thứ sáu ngày 13 xảy ra. Hàm `M_file` sẽ cho bạn những thông tin này.

```
function m=friday(start)
```

```
% FRIDAY Date of the next Friday the 13th
```

```
% FRIDA ' display the next occurrence of Friday the
```

```
% 13th
```

```
% FRIDAY(START) start the search at the date
```

```

        % specified by START
    % M=FRIDAY return the date number of the next Friday
        % the 13th
    if nargin==0
        start=now;      % use the current date if none
    end                % was supplied
    [yr,mo,da]=datevec(start);
    da=da+6-weekday(start); % Start with the Friday in
                            % this week
    start=datenum(yr,mo,da,0,0,0);
    while 1
        [yr,mo,da]=datevec(start);
        if (weekday(start)==6)&(da==13)
            break;
        end
        start=datenum(start+7); % skip to the next Friday
    end
    if nargout==0
        disp(['Friday,'datestr(start,1)]) % Display the
                                           % the result
    else
        m=start;      % or return the resulting date
    end                % number

```

Sau khi chạy chương trình ta được kết quả:

```
>> friday
```

```
Friday, 13-Aug-1999
```

Nếu bạn muốn được cảnh báo cho toàn bộ năm, xem hàm `fridays`:


```

function F=fridays(ynum)
% FRIDAY List the Friday the 13ths in the year ynum.
% M=FRIDAY return the date numbers found.
%
if nargin==0
    [ynum dummy]=datevec(now); % use the current date if
end % none was supplied
MM=[];
trynum=datetime(ynum,1,13,0,0,0);
    % check January 13 first
trynum=friday(trynum); % find the first one
[tyr dummy]=datevec(trynum);
while tyr==ynum % May be there are more this year
    MM=[MM;trynum];
    trynum=friday(trynum+7); % skip to the next week
    [tyr dummy]=datevec(trynum);
end
if nargin==0
    disp('Fridays'); % Display the results
    disp(datestr(MM,1)) % Display the result
else
    F=MM; % or return the vector of
end % date number

```

VÒNG LẶP ĐIỀU KHIỂN

Các ngôn ngữ lập trình và máy tính có khả năng lập trình đều cho phép bạn điều khiển vòng lặp của các câu lệnh dựa trên những cấu trúc của nó. Nếu bạn đã từng sử dụng những ứng dụng này thì phần này sẽ rất đơn giản đối với bạn. Mặt khác nếu vòng lặp điều khiển là mới đối với bạn thì nó sẽ rất rắc rối, nếu như vậy, thì bạn hãy nghỉ ngơi chút nữa rồi tiếp tục.

Vòng lặp điều khiển rất hữu ích và có ứng dụng rất rộng rãi, nó làm cho các phép toán được thực hiện một cách thuận tiện hơn và nhanh hơn. MATLAB đưa ra các dạng vòng lặp có điều khiển là: vòng lặp *for*, vòng lặp *while*, cấu trúc *if-else-end* và cấu trúc *switch-case*. Vì các cấu trúc thường hoàn thiện các lệnh của MATLAB, nên chúng thường xuất hiện trong M_file, hơn là trong câu lệnh đánh trực tiếp tại dấu nhắc của MATLAB.

11.1 Vòng lặp *for*

Vòng lặp *for* cho phép một nhóm lệnh thực hiện lặp lại một số lần cố định. Cú pháp của vòng lặp *for* như sau:

```
for x = array
    commands    % Khối các lệnh
end
```

Các câu lệnh giữa hai trạng thái *for* và *end* được thực hiện một lần cho tất cả các cột của mảng (array). Tại mỗi lần lặp lại, x được gán cho phần tử cột tiếp theo như trong suốt n lần của vòng lặp, $x = \text{array}(:, n)$.

Ví dụ:

```
>> for n = 1:10
    x(n) = sin(n*pi/10)
end
```

```
>> x
```

```
x =
```

```
Columns 1 through 7
```

```
0.3090 0.5878 0.8090 0.9511 1.0000 0.9511 0.8090
```

```
Columns 8 through 10
```

```
0.5878 0.3090 0.0000
```

Nói một cách khác, trạng thái thứ nhất yêu cầu: Cho n bằng từ 1 đến 10, tính giá trị của tất cả các trạng thái cho đến trạng thái kế tiếp trạng thái *end*. Đầu tiên trong vòng lặp *for* $n=1$, tiếp theo $n=2$, và cứ như vậy cho đến trường hợp $n=10$. Sau trường hợp $n=10$, vòng lặp *for* kết thúc, và tất cả các lệnh sau trạng thái *end* của vòng lặp được thực hiện.

Vòng lặp *for* không thể bị kết thúc bằng cách gán lại biến điều khiển n trong vòng lặp:

```
>> for n = 1:10
```

```
    x(n) = sin(n*pi/10);
```

```
    n = 10;
```

```
end
```

```
>> x
```

```
x =
```

```
Columns 1 through 7
```

```
0.3090 0.5878 0.8090 0.9511 1.0000 0.9511 0.8090
```

```
Columns 8 through 10
```

```
0.5878 0.3090 0.0000
```

Trạng thái 1:10 là một trạng thái tạo lên mảng MATLAB tiêu chuẩn. Bất cứ kiểu mảng nào của MATLAB đều được chấp nhận trong vòng lặp *for*.

```
>> data = [3 9 45 6; 7 16 -1 5]
```

```
data =
```

```
3    9   45    6
```

```
7   16   -1    5
```

```
>> for n = data
```

```
    x = n(1)-n(2)
```

```

end
x =
    -4
x =
    -7
x =
    46
x =
     1

```

Bình thường vòng lặp *for* có thể lồng vào nhau:

```

>> for n = 1:5
    for m = 5:-1:1
        A(n,m) = n^2+m^2;
    end
    disp(n)
end
1
2
3
4
5
>> A
A =
     2     5    10    17    26
     5     8    13    20    29
    10    13    18    25    34
    17    20    25    32    41
    26    29    34    41    50

```

Không nên dùng vòng lặp *for* khi mà tương đương với việc ta dùng mảng để tính toán. Như trong ví dụ trước ta cũng có thể dùng mảng để tính toán:

```
>> n = 1: 10;  
>> x = sin(n*pi/10)  
x =  
Columns 1 through 7  
0.3090 0.5878 0.8090 0.9511 1.0000 0.9511 0.8090  
Columns 8 through 10  
0.5878 0.3090 0.0000
```

Trong hai trường hợp như trên, trường hợp thứ hai ta dùng mảng để tính toán cũng được kết quả như vậy, nhưng nó nhanh hơn và các thao tác cũng ít hơn.

Để tăng tốc độ tính toán, mảng cần phải được khởi tạo trước khi thực hiện vòng lặp *for* (hoặc vòng lặp *while*). Trong ví dụ trước cứ mỗi lần lệnh trong vòng lặp *for* được tính, kích cỡ của biến *x* lại tăng lên 1. Điều này làm cho MATLAB mất thời gian để cập nhật thêm bộ nhớ cho *x* trong mỗi vòng. Để rút ngắn bước này, ví dụ về vòng lặp *for* ở trước viết lại như sau:

```
>> x = zeros(1,10); % Khởi tạo bộ nhớ cho x  
>> for n = 1: 10  
    x = sin(n*pi/10);  
end
```

Bây giờ chỉ cần thay đổi giá trị của các phần tử của *x*.

11.2 Vòng lặp while

Vòng lặp *while* thực hiện lặp lại một nhóm lệnh một số lần cố định, nhưng không biết trước được số lần lặp lại.

Cú pháp của vòng lặp *while* như sau:

```
while biểu thức điều kiện  
    khối các lệnh..  
end
```

"khối các lệnh.." giữa hai trạng thái *while* và *end* được thực hiện lặp đi lặp lại khi tất cả các "biểu thức điều kiện" là đúng. Thông thường giá trị của điều kiện đưa ra kết quả là một số, nhưng

nếu các kết quả đưa ra là một mảng thì vẫn hợp lệ. Trong trường hợp mảng, tất cả các phần tử trong mảng kết quả đưa ra phải là True (đúng). Có thể tham khảo ví dụ dưới đây:

```
>> num = 0, ESP = 1,  
>> while (1+ESP) > 1  
    ESP = ESP/ 2;  
    num = num + 1;  
  
end  
  
>> num  
num=  
    53  
  
>> ESP = 2*ESP  
ESP=  
    2.2204e-16
```

Ví dụ này đưa ra cách tính giá trị đặc biệt eps của MATLAB, nó là một số dương nhỏ nhất, có thể cộng với 1 để được một số lớn hơn 1 dùng cho giới hạn độ chính xác. Ở đây chúng ta dùng chữ hoa EPS để chắc chắn rằng giá trị eps của MATLAB không ghi đè lên. Trong ví dụ này, giá trị của EPS bắt đầu bằng 1, trong khi điều kiện $(1+EPS) > 1$ là True (để cho nó khác không), các lệnh trong vòng lặp *while* được tính, giá trị của EPS tiếp tục được chia đôi, giá trị của EPS nhỏ đi, mà cộng EPS với 1 thì nó là số nhỏ nhất mà lớn hơn 1. Do máy tính sử dụng số cố định có 16 chữ số nên khi giá trị nhỏ quá thì nó làm tròn bằng 0, và khi đó điều kiện $(EPS+1) > 1$ False (sai) và vòng lặp *while* dừng lại. Cuối cùng EPS được nhân với 2 vì sau lần chia cuối cùng cho 2 thì vòng lặp dừng lại.

11.3 Cấu trúc if-else-end

Nhiều khi chúng ta cần những câu lệnh được thực hiện theo một điều kiện nào đó. Trong ngôn ngữ lập trình, logic này được cung cấp bởi cấu trúc *if-else-end*. Cú pháp của cấu trúc này như sau:

```
if biểu thức điều kiện  
    khối các lệnh. .  
  
end
```

Khối các lệnh giữa hai trạng thái *if* và *end* được thực hiện khi tất biểu thức điều kiện là đúng. Trong trường hợp điều kiện bao gồm các điều kiện con, thì tất cả các điều kiện con được tính và trả về một trạng thái logic của điều kiện. Ví dụ:

```

>> apple = 10          % số táo
>> cost = apple*25
cost=
    250
>> if apple > 5
    cost = (1-20/100)*cost;    % bỏ đi 20%
end
>> cost
cost
    200

```

Trong trường hợp có hai điều kiện thay đổi, cấu trúc *if-else-end* là:

```

if biểu thức điều kiện
    khối các lệnh được thực hiện nếu điều kiện là đúng
else
    khối các lệnh được thực hiện nếu điều kiện là sai
end

```

Khi có ba hoặc nhiều điều kiện thay đổi, cấu trúc của nó sẽ là:

```

if biểu thức điều kiện 1
    khối: các lệnh được thực hiện nếu điều kiện 1 là đúng
elseif biểu thức điều kiện 2
    khối các lệnh được thực hiện nếu điều kiện 2 là đúng
elseif biểu thức điều kiện 3
    khối các lệnh được thực hiện nếu điều kiện 3 là đúng
elseif biểu thức điều kiện 4

```

```

else

```

khỏi các lệnh được thực hiện nếu không có điều kiện nào đúng.

end

Trong mẫu dạng này thì khi biểu thức điều kiện đầu tiên đúng thì các câu lệnh sau không được kiểm tra nữa, các cấu trúc **if-else-end** còn lại được bỏ qua. Hơn nữa câu lệnh **else** ở cuối có thể không cần cho vào.

Đối với cấu trúc **if-else-end**, chúng ta cũng có thể lồng vào các vòng lặp **for** và **while**:

```
>> EPS = 1;
>> for num = 1:100
    EPS = EPS/ 2;
    if (1+EPS)< 1
        EPS = EPS*2
        break
    end
end
EPS =
    2.2204e-16
>> num
num=
    53
```

Ví dụ này đưa ra cách khác để tính số eps. Trong ví dụ, khi lệnh **break** được thực hiện thì MATLAB nhảy ra khỏi vòng lặp nó đang thực hiện. Khi lệnh **break** xuất hiện trong một vòng lặp **for** hoặc **while** trong các vòng lặp lồng nhau thì nó chỉ nhảy ra khỏi một vòng lặp chứa nó chứ nó không nhảy ra khỏi tất cả các vòng lặp.

11.4 Cấu trúc switch-case

Khi một chuỗi các lệnh đánh giá dựa trên một biểu thức thứ hoặc biểu thức điều kiện với nhiều giá trị thứ khác nhau, người ta thường dùng cấu trúc **switch-case**. Cấu trúc **switch-case** có dạng như sau:

switch biểu thức điều kiện
 case giá trị thứ 1


```

        khối lệnh 1
    case { giá trị thứ 2, giá trị thứ 3, giá trị thứ 4}
        khối lệnh 2
    otherwise
        khối lệnh 3
end

```

Tại đây biểu thức điều kiện phải là dạng số hoặc dạng chuỗi, nếu biểu thức điều kiện là dạng số thì lệnh **case** sẽ thử xem giá trị của biểu thức đó có bằng giá trị thứ *i* hay không. Nếu biểu thức điều kiện là một chuỗi thì lệnh **case** sẽ so sánh chuỗi đó với giá trị thứ *i*. Trong ví dụ trước, biểu thức điều kiện được đem so sánh với giá trị thứ 1, nếu chúng bằng nhau thì khối lệnh đầu tiên được thực hiện, mà các khối lệnh tiếp theo cho đến trước trạng thái **end** được bỏ qua, nếu chúng không bằng nhau thì điều kiện tiếp tục được đem so sánh với giá trị thứ 2, giá trị thứ 3, giá trị thứ 4, nếu một trong các giá trị này bằng biểu thức điều kiện thì khối lệnh 2 được thực hiện. Nếu tất cả các lệnh so sánh của **case** đều không đúng thì khối lệnh 3 được thực hiện. Chú ý rằng trong cấu trúc **switch-case** có ít nhất một nhóm lệnh phải được thực hiện. Sau đây là một ví dụ về cấu trúc **switch-case**.

```

x = 2.7;
units = 'm';
switch units      % Chuyển x ra centimeters
case {'inch','in'}
    y=x*2.54;
case {'feet','ft'}
    y=x*2.54*12;
case {'meter','m'}
    y=x/ 100;
case {'millimeter','mm'}
    y=x*10;
case {'centimeter','cm'}
    y=x;
otherwise

```

```

disp(['không biết units: ' units])
y=nan;

end

```

Khi thực hiện ví dụ này thì giá trị cuối cùng của y là: y=0.027.

Ví dụ: Vấn đề về lãi suất

Vấn đề: Để mua một ô tô, Dũng phải vay 10,000\$ với lãi suất hàng tháng là 8.9%. Trong 3 năm gốc và lãi được tính như thế nào sau mỗi lần chi trả. Ngoài ra phần tiền còn lại sau mỗi lần chi trả là bao nhiêu?

Giải pháp: Từ chương 2 số tiền chi trả P hàng tháng cho khoản vay A dollar với lãi suất hàng tháng là R, tính trong M tháng là:

$$P = A \cdot \left[\frac{R(1+R)^M}{(1+R)^M - 1} \right]$$

Tại lần chi trả đầu tiên, tiền lãi phải trả là $I_{p1} = R \cdot A$. Giả sử số tiền phải trả là P thì tiền gốc phải trả là $P_{r1} = P - I_{p1}$ và số tiền còn lại sau lần chi trả thứ nhất là $B_1 = A - P_{r1}$. Trong tất cả các lần chi trả sau đó tiền lãi phải trả là $I_{pm} = R \cdot B_{m-1}$ và số tiền còn lại là $B_m = B_{m-1} - P_{rm}$. Sử dụng các thông tin này thì chương trình MATLAB sẽ như sau:

```

function amort

% amort.m script file

A=10000; % amount of loan

M=3*12; % number of months

R=8.9; % annual interest rate

r=(R/100)/12; % monthly interest rate

P=A*(r*(1+r)^M/((1+r)^M-1)); % payment required

B=zeros(M,1); % storage for balance remaining per month

Ip=B; % storage for interest paid per month

Pr=B; % storage for principle paid per month

for m=1:M

    if m==1 % compute interest when balance is
        Ip(m)=r*A; % original amount
    end
end

```

```

else
    Ip(m)=r*B(m-1);
end
Pr(m)=P-Ip(m); % principle paid this month
if m==1 % compute balance remaining after payment
    B(m)=A-Pr(m);
else
    B(m)=B(m-1)-Pr(m);
end
end
format bank
disp(['Amount=' num2str(A)])
disp(['Interest Rate=' num2str(R)])
disp(['Number of months = ' num2str(M)])
disp(['Payment =' num2str(P)])
disp(' ')
disp('          Amortization Schedule')
disp(' Payment Balance Interest Principle')
disp([(1:M)' B Ip Pr'])
format short g

```

Chạy chương trình này thì kết quả như sau:

```
>> Amount=10000
```

```
Interest Rate=8.9
```

```
Number of months = 36
```

```
Payment =317.5321
```

```
          Amortization Schedule
```

Payment	Balance	Interest	Principle
1.00	9756.63	74.17	243.37

2.00	9511.46	72.36	245.17
3.00	9264.48	70.54	246.99
4.00	9015.65	68.71	248.82
5.00	8764.99	66.87	250.67
6.00	8512.46	65.01	252.53
7.00	8258.07	63.13	254.40
8.00	8001.78	61.25	256.28
9.00	7743.60	59.35	258.19
10.00	7483.49	57.43	260.10
11.00	7221.47	55.50	262.03
12.00	6957.49	53.56	263.97
13.00	6691.56	51.60	265.93
14.00	6423.66	49.63	267.90
15.00	6153.77	47.64	269.89
16.00	5881.88	45.64	271.89
17.00	5607.97	43.62	273.91
18.00	5332.03	41.59	275.94
19.00	5054.04	39.55	277.99
20.00	4773.99	37.48	280.05
21.00	4491.87	35.41	282.12
22.00	4207.65	33.31	284.22
23.00	3921.33	31.21	286.33
24.00	3632.88	29.08	288.45
25.00	3342.29	26.94	290.59
26.00	3049.55	24.79	292.74
27.00	2754.63	22.62	294.91
28.00	2457.53	20.43	297.10
29.00	2158.22	18.23	299.31

30.00	1856.70	16.01	301.53
31.00	1552.94	13.77	303.76
32.00	1246.92	11.52	306.01
33.00	938.64	9.25	308.28
34.00	628.07	6.96	310.57
35.00	315.19	4.66	312.87
36.00	-0.00	2.34	315.19

Ví dụ này minh họa cấu trúc lặp *for* và *if-else-end*. Nó cũng minh họa việc sử dụng script M_file. Để tính toán một khoản cho vay bất kỳ bạn chỉ cần thay đổi dữ liệu vào ở phần đầu của chương trình và bạn chạy lại nó.

Ví dụ: Chuỗi lên xuống

Vấn đề: Cho x_0 là một số nguyên bất kỳ. Giả sử chuỗi x_k được định nghĩa như sau:

$$x_{k+1} = x_k / 2 \quad \text{nếu } x_k \text{ là chẵn}$$

$$\text{và} \quad x_{k+1} = 3x_k + 1 \quad \text{nếu } x_k \text{ là lẻ}$$

Chuỗi này có thuộc tính gì nếu chuỗi số dừng lại khi $x_k = 1$, chuỗi phân kỳ hay hội tụ về 1.

Giải pháp: Chúng ta chỉ cần vòng lặp *while* để xét xem khi nào $x_k = 1$ và sử dụng cấu trúc

if-else-end để thực hiện việc tính toán dãy x_k . Trong MATLAB thì chương trình như sau:

```
function up_down
```

```
% up_down.m script file for up/down sequence problem
```

```
x=zeros(500,1); %preallocate storage for x(k)
```

```
x(1)=round(abs(input('Enter a number> ')));
```

```
k=1;
```

```
while (x(k)>1)&(k<500)
```

```
    if rem(x(k),2)==0 % x(k) is even
```

```
        x(k+1)=x(k)/2;
```

```
    else % x(k) is odd
```

```
        x(k+1)=3*x(k)+1;
```

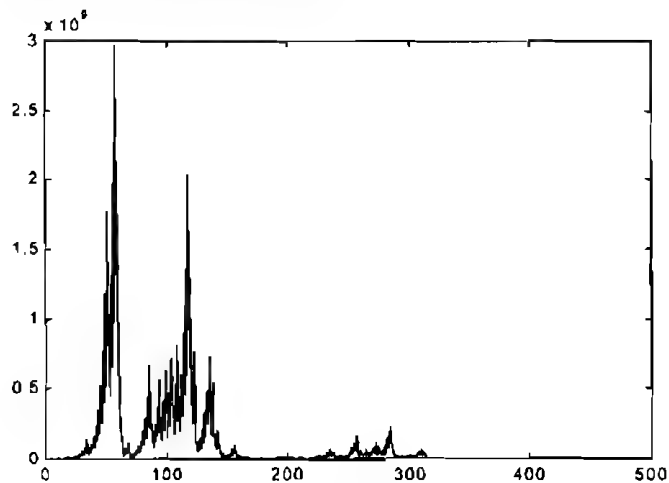
```

end
k=k+1, % increment sequence counter
end
x=x(x>0) % keep values generated only and display them
M=0.499;
plot(M,x)

```

Kết quả của chương trình này khá thú vị, ví dụ với $x=2^m$, trong đó m là một số nguyên thì chuỗi sẽ rất ngắn (tại sao?), hơn nữa bất cứ khi nào giá trị của một số hạng trong chuỗi là lũy thừa của 2 thì chuỗi sẽ nhanh chóng dừng lại, nhưng đối với những số x tương đối nhỏ thì kết quả là một chuỗi khá thú vị. Ví dụ $x1=27$. Hầu như tất cả các giá trị ban đầu đều sinh ra một chuỗi có giá trị rất ngẫu nhiên như hình vẽ dưới đây với $x(1)=837799$. Liệu bạn có dám kết luận chuỗi này hội tụ hay không!

Đồ thị kết quả của chương trình với $x(1)=837799$ là:



Hình 11.1

HÀM M_FILE

Khi bạn sử dụng các hàm MATLAB như *inv*, *abs*, *angle*, và *sqr*t, MATLAB nhận giá trị mà bạn truyền vào, dựa vào kết quả đó, tính toán kết quả của hàm và trả lại cho bạn kết quả tính toán. Các lệnh tính toán bằng hàm cũng như các biến trung gian được tạo ra bởi các lệnh này bạn đều không nhìn thấy, tất cả những gì bạn trông thấy chỉ là các giá trị nhập vào và các giá trị đưa ra, vì vậy có thể coi một hàm như một cái hộp đen. Các thuộc tính này làm cho hàm trở lên rất hữu dụng đối với các lệnh tính toán mà phải dùng đến các hàm toán học phức tạp thường xuất hiện khi bạn giải quyết những vấn đề lớn. Dựa vào ưu điểm này, MATLAB cung cấp một cấu trúc để bạn có thể tự tạo một hàm cho mình dưới dạng một M_file. Hàm *flipup* dưới đây là một ví dụ về việc dùng hàm M_file:

```
function y=flipup(x)

% FLIPUP Flip matrix in up/down direction.

% FLIPUP(x) return x with columns preserved and rows flipped

% in the up/down direction. For example.

%
% x = 1 4      becomes 3 6
%      2 5          2 5
%      3 6          1 4
%
% See also FLIPLR, ROT90, FLIPDIM.
% Copyright (c) 1984-96 by the MathWorks, Inc.
% $Revision: 5.3 $ $Date: 1996/10/24 18:41:14 $

if ndim(x)~=2
    error('X must be a 2-D matrix.');
```

```

end

[m n] = size(x);

y = x(m:-1:1,:);

```

Một hàm M_file rất giống với một script file bởi vì chúng cùng là các file văn bản và cùng có phần mở rộng là '.m'. Điểm khác nhau giữa script file và các hàm M_file là các hàm M_file không được nhập vào từ cửa sổ lệnh mà thông qua một trình soạn thảo văn bản từ bên ngoài. Hàm M_file còn khác với script file ở chỗ nó chỉ thông tin với MATLAB thông qua các biến truyền vào cho nó và thông qua các biến ra mà nó tạo thành, các biến trung gian ở bên trong hàm thì không xuất hiện hay tương tác với môi trường của MATLAB. Như bạn có thể thấy ở ví dụ trước, dòng đầu tiên của hàm M_file định nghĩa file này như một hàm và chỉ ra tên của nó, tên này chính là tên file nhưng không có phần mở rộng là '.m' đồng thời nó cũng định nghĩa luôn biến vào và ra. Chuỗi các dòng lệnh tiếp theo là các lời chú thích, sẽ xuất hiện khi ta dùng lệnh `>>help`, `>>help flipud`, hoặc `>>helpw nflipud`, dòng lệnh `help` đầu tiên gọi là dòng H1 chính là dòng hiện ra khi dùng lệnh `lookfor`. Cuối cùng phần còn lại của file này chứa các lệnh của MATLAB để tạo lên các biến ra.

12.1 Các quy luật và thuộc tính

Hàm M_file phải tuân theo những quy luật và thuộc tính nhất định, ngoài ra chúng còn có một số tính chất rất quan trọng bao gồm:

- Tên hàm và tên file phải là một, ví dụ hàm *flipud* phải được lưu trong file với cái tên là *flipud.m*.
- Khi MATLAB thực hiện lần đầu hàm M_file, nó sẽ mở file văn bản tương ứng và dịch các dòng lệnh của file đó ra một dạng mã lưu trong bộ nhớ nhằm mục đích tăng tốc độ thực hiện các lời gọi hàm tiếp theo. Nếu trong hàm có chứa lời gọi hàm M_file khác thì các hàm đó cũng được dịch vào trong bộ nhớ.
- Các dòng ghi lời chú thích cho tới dòng đầu tiên không phải là chú thích trong hàm M_file là những dòng văn bản, nó sẽ hiện ra khi bạn sử dụng lệnh *help*. Ví dụ `>>help flipud` sẽ trả về 9 dòng đầu tiên trong hàm M_file nói trên. Dòng đầu tiên là dòng H1, nó sẽ xuất hiện khi bạn dùng lệnh *look for*.
- Mỗi hàm có một không gian làm việc riêng tách biệt so với môi trường MATLAB, mối quan hệ duy nhất giữa các biến trong hàm với môi trường MATLAB là các biến vào và ra của hàm đó. Nếu bản thân hàm giá trị bị thay đổi thì sự thay đổi này chỉ tác động bên trong của hàm đó mà không làm ảnh hưởng đến các biến của môi trường MATLAB. Các biến được tạo ra bên trong một hàm thì chỉ nằm trong không gian làm việc của hàm đó và được giải phóng khi hàm kết thúc, vì vậy không thể sử dụng thông tin của lần gọi trước cho lần gọi sau.

- Số các tham số vào và ra khi một hàm được gọi thì chỉ có tác dụng bên trong hàm đó bên nargin chứa các tham số đưa vào còn biến nargin chứa các giá trị đưa ra, trong thực tế thì các biến này thường được sử dụng để xác định giá trị ra dựa vào số lượng các đối số đưa vào. Ví dụ xét hàm linspace sau:

```
function y=linspace(d1, d2, n)
% LINESPACE Linearly spaced vector.
% LINESPACE(x1, x2) generates a row vector of 100 linearly
% equally spaced points between x1 and x2.
%
% LINESPACE(x1, x2, N) generates N points between x1 and x2.
%
% See also LOGSPACE,;.
% Copyright (c) 1984-96 by the MathWorks, Inc.
% $Revision: 5.3 $ $Date: 1996/10/24 18:41:14 $
if nargin==2
    n = 100;
end
y = [d1 + (0:n-2)*(d2-d1)/(n-1) d2];
```

Ở đây nếu lời gọi của người sử dụng chỉ truyền vào hai đối số thì linspace trả về giá trị 100, nhưng nếu số đối số là 3, ví dụ như linspace(0,10,50) thì đối số thứ 3 sẽ quyết định số các điểm dữ liệu.

- Các hàm có thể dùng chung các biến với hàm khác, với môi trường MATLAB và có thể đệ quy nếu như các biến được khai báo là toàn cục. Để có thể truy cập đến các biến trong một hàm hoặc trong môi trường MATLAB thì các biến đó phải được khai báo là biến toàn cục trong mỗi hàm sử dụng nó. Hàm *tic* và *toc* sau đây mô tả một ví dụ về việc sử dụng biến toàn cục:

```
function tic
% TIC Start a stopwatch timer.
% The sequence of lệnhs
% TIC, operation, TOC
```

```

% prints the time required for the operation.
%
% See also TOC, CLOCK, ETIME, CPUTIME.
% Copyright (c) 1984-96 by the MathWork, Inc.
% $Revision: 5.3 $ $Date: 1996/10/24 18:41:14 $
% TIC simple stores CLOCK in a global variable
global TICTOC
TICTOC = clock;
function t = toc
% TOC Read the stopwatch timer.
% TOC, by itself, prints the elapsed time in t,
% instead of printing it out.
%
% See also TIC, ETIME, CLOCK, CPUTIME.
% Copyright (c) 1984-96 by the MathWork, Inc.
% $Revision: 5.3 $ $Date: 1996/10/24 18:41:14 $
% TOC uses ETIME and the value of clock saved by TIC.
global TICTOC
if nargin < 1
    elapsed_time = etime(clock, TICTOC);
else
    t = etime(clock, TICTOC);
end

```

Trong hàm *tic* thì biến TICTOC được khai báo là biến toàn cục và giá trị của biến này có được thông qua việc gọi hàm *clock*. Sau đó trong hàm *toc*, biến TICTOC cũng được khai báo là biến toàn cục làm cho *toc* có khả năng truy cập đến biến TICTOC ở trong hàm *tic*, sử dụng giá trị của biến này *toc* sẽ tính được khoảng thời gian đã trôi qua kể từ khi hàm *tic* được thi hành. Một điều quan trọng cần nhớ là biến TICTOC chỉ tồn tại trong không gian làm việc của *tic* và *toc* nhưng không tồn tại trong môi trường MATLAB.

- Việc thi hành hàm `M_file` sẽ kết thúc khi gặp dòng cuối cùng của file đó hoặc gặp dòng lệnh ***return***. Lệnh ***return*** giúp ta kết thúc một hàm mà không cần phải thi hành hết các lệnh của hàm đó.

- Hàm ***error*** của MATLAB sẽ hiển thị một chuỗi tên của số lệnh và dừng thực hiện hàm, trả điều khiển về cho cửa sổ lệnh và bàn phím. Hàm này rất hữu dụng để cảnh báo việc sử dụng hàm không đúng mục đích. Ví dụ như câu lệnh sau

```
if length(val) > 1
    error('VAL phải là giá trị số!')
end
```

Ở đây nếu `val` không phải là số thì hàm ***error*** sẽ hiện lên chuỗi cảnh báo và trả điều khiển cho cửa sổ lệnh và bàn phím.

- Một `M_file` có thể chứa nhiều hàm. Hàm chính trong `M_file` này phải được đặt tên trùng với tên của `M_file` như đề cập đến ở trên. Các hàm khác được khai báo thông qua câu lệnh ***function*** được viết sau hàm đầu tiên. Các hàm con chỉ được sử dụng bởi hàm chính, có nghĩa là ngoài hàm chính ra thì không có hàm nào khác có thể gọi được chúng. Tính năng này cung cấp một giải pháp hữu hiệu để giải quyết từng phần của hàm chính một cách riêng rẽ làm giảm bớt các khó khăn khi ta lập trình một hàm lớn.

Nói tóm lại, hàm `M_file` cung cấp cho ta một phương pháp đơn giản để mở rộng khả năng của MATLAB. Trong thực tế rất nhiều hàm của MATLAB là các hàm `M_file`.

12.2. Các ví dụ

Ví dụ: Hàm trả dần theo thời hạn

Vấn đề: Giả sử có một khoản cho vay A dollar, với lãi suất hàng tháng là $R\%$ và phải trả trong vòng M tháng. Hãy viết một hàm `M_file` để thể hiện:

- Lịch chi trả nếu như ban đầu chưa biết các số liệu đưa ra
- Số tiền chi trả hàng tháng nếu biết một số liệu ra.
- Số tiền chi trả hàng tháng và một ma trận số chứa lịch thanh toán nếu biết trước hai đối số ra.

Giải pháp: Trong chương 2, số tiền phải chi trả hàng tháng P cho khoản cho vay A dollar với lãi suất là R , trả trong M tháng:

$$P = A \cdot \left[\frac{R(1+R)^M}{(1+R)^M - 1} \right]$$

Tại lần chi trả đầu tiên, tiền lãi phải trả là $I_{p1} = R.A$. Giả sử số tiền phải trả là P thì tiền gốc phải trả là $P_{r1} = P - I_{p1}$ và số tiền còn lại sau lần chi trả thứ nhất là $B_1 = A - P_{r1}$. Trong tất cả các lần chi trả sau đó tiền lãi phải trả là $I_{pm} = R.B_{m-1}$ và số tiền còn lại là $B_m = B_{m-1} - P_{rm}$. Sử dụng các thông tin này thì chương trình MATLAB sẽ như sau:

```
function [P,S]=loan(a,r,m)

%LOAN Loan Payment and Amortization Table.

% (H1 help line)

%P=LOAN(A,R,M) computes the monthly payment on a loan

%amount of a, having an annual interest rate of R,

% to be paid off in equal amounts over M months.

%

%[P,S]=LOAN(A,R,M) also returns

% an amortization table S,

%which is an M-by-4 matrix

% where S(:,1)=Payment Number,

%S(:,2)=Remaining Balance, S(:,3)=Interest Paid, and

%S(:,4)=Principle Paid.

%

%If no output arguments are provided

% the table is displayed.

%Start with some error checking

if nargin<3

    error('Three input argument are required.')

end

if fix(m)~=m

    error('Number of Months Must be Integer.')

end

    % Now calculate

rm=(r/100)/12;          % Monthly interest rate
```

```

p=a*(rm*(1+rm)^m/((1+rm)^m-1)); % payment required
if nargout==1      % done if only payment is required.
    P=p;           % copy out into output variable
    return
end
B=zeros(m,1);    % storage for balance remaining per month
lp=B;            % storage for interest paid per month
Pr=B;            % storage for principal paid per month
for i=1:m        % creat table data
    if i==1
        % compute interest when balance is orginnal amount
        lp(i)=rm*a;
    else          % balance is B(i-1)
        lp(i)=rm*B(i-1);
    end
    Pr(i)=p-lp(i); %principal paid this month
    if i==1      % compute balance remainig after payment
        B(i)=a-Pr(i);
    else
        B(i)=B(i-1)-Pr(i);
    end
end
end
B(abs(B)<0.001)=0; % set near zero balance to zero
s=[(1:m)' B lp Pr];
if nargout==0    % display table
    disp(['Amount = ' num2str(a)])
    disp(['Interest rate = ' num2str(r)])
    disp(['Number of month = ' int2str(m)])

```

```

disp(['Payment = ' num2str(p)])
disp(' ')
disp('      Amortization Schedule')
disp(' Payment Balance Interest Principle')
fprintf(' %5.0f %12.2f %12.2f %12.2f\n', s)

        % better formatting
else    % two output arguments requested
    P=p;
    S=s;
end

```

Ví dụ: Giải mã màu trên các điện trở

Vấn đề: Giá trị của một điện trở dung trong mạch điện được tính thông qua các vạch màu in trên thân của nó. Đối với một điện trở với độ chính xác là 5% thì có 3 dải màu, tạm gọi là A, B, C. Giá trị số được gán cho mỗi màu được tính như sau:

Màu	Giá trị tương ứng
Đen	0
Nâu	1
Đỏ	2
Đa cam	3
Vàng	4
Xanh lá cây	5
Xanh da trời	6
Tím	7
Xám	8
Trắng	9

Nếu A, B, C là các giá trị của các màu trên giải mã thì giá trị của các điện trở là:

$$R = (10.A + B).10^C$$

Sử dụng các thông tin này, hãy tạo một M-file trả về giá trị của điện trở ứng với bất kỳ một điện trở chuẩn nào.

Giải pháp: Vấn đề này yêu cầu một chuỗi các thao tác và so sánh để thực hiện sự chuyển đổi trong bảng trên. Giải pháp của MATLAB là:

```
function r=resistor(a, b, c)
%RESISTOR(A, B, C) Resistor value from color code.
%RESISTOR(a, B, C) returns the resistance
%value of resistor
%given its three color bands, A, B, C.
%A, B, C must be one of the
%following character strings:
%
%'black', 'brown', 'red', 'orange', 'yellow',
%'green', 'blue', 'violet', 'gray', 'white'
% first some error checking
if nargin~=3
    error('Three input arguments required')
end
if ~ischar(a)|~ischar(b)|~ischar(c)
    error('Inputs Must be Character Xáus')
end

                                % now solve problem
vals=zeros(1,3);              %~string cell array of three inputs
abc={a,b,c};                  % tring cell array ó thrê input
for i=1:3                      % do each color band in turn
    band=lower(abc(i));
                                % get (i)th input and make lower case
    if strcmp(band,'bla',3) % black (compare min # of)
```

```

        vals(i)=0;      % chars for unique match)
elseif strcmp(band,'br',2) %brown
    vals(i)=1;
elseif strcmp(band,'r',1) %red
    vals(i)=2;
elseif strcmp(band,'o',1) %orange
    vals(i)=3;
elseif strcmp(band,'y',1) %yellow
    vals(i)=4;
elseif strcmp(band,'gre',3) %green
    vals(i)=5;
elseif strcmp(band,'blu',3) %blue
    vals(i)=6;
elseif strcmp(band,'v',1) %violet
    vals(i)=7;
elseif strcmp(band,'gra',3) %gray
    vals(i)=8;
elseif strcmp(band,'w',1) %white
    vals(i)=9;
else
    error(['Unknown Color Band.'])
end
end

if vals(1)==0
    error('First Color Band Cannot Be Black.')
end

r=(10*vals(1)+vals(2))*10^vals(3);

```


Sử dụng hàm này cho một vài ví dụ

```
>> resistor('brown', 'black', 'red')
```

```
ans=
```

```
1000
```

PHÂN TÍCH DỮ LIỆU

MATLAB là một chương trình ứng dụng hướng ma trận nên nó dễ dàng thực hiện các phân tích thống kê trên các tập dữ liệu, trong khi theo mặc định MATLAB coi các tập dữ liệu được lưu trữ trong các mảng cột, việc phân tích dữ liệu có thể thực hiện theo bất cứ chiều nào. Đó là trừ khi được chỉ định theo một cách khác, các cột của một mảng dữ liệu thể hiện các thông số đo khác nhau, mỗi hàng thể hiện một giá trị mẫu của các thông số đo đó. Ví dụ giả sử nhiệt độ ban ngày (tính theo độ C) của 3 thành phố tính trong một tháng (31 ngày được ghi lại và gán cho một biến là temps trong một script M_file, khi chạy M_file thì giá trị của temps được đưa vào môi trường MATLAB, thực hiện công việc này, biến temps chứa:

```
>> temps
```

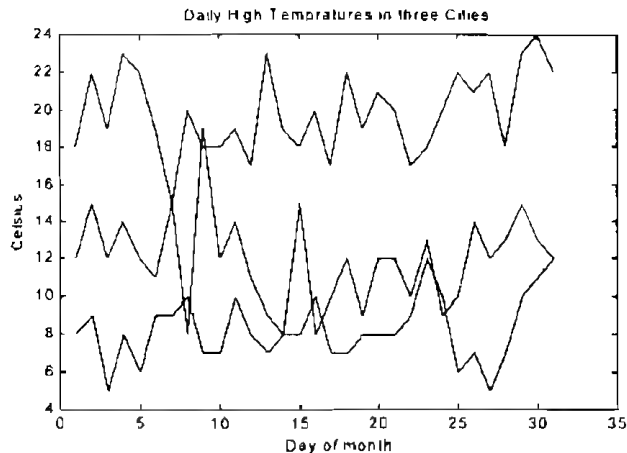
```
temps=
```

```
12    8    18
15    9    22
12    5    19
14    8    23
12    6    22
11    9    19
15    9    15
 8   10    20
19    7    18
12    7    18
14   10    19
```

11	8	17
9	7	23
8	8	19
15	8	18
8	10	20
10	7	17
12	7	22
9	8	19
12	8	21
12	8	20
10	9	17
13	12	18
9	10	20
10	6	22
14	7	21
12	5	22
13	7	18
15	10	23
13	11	24
12	12	22

Mỗi hàng chứa nhiệt độ của một ngày nào đó, còn mỗi cột chứa nhiệt độ của một thành phố. Để cho dữ liệu trở lên dễ dàng hơn, hãy gõ vào như sau:

```
>> d=1:31; % number the days of the month
>> plot(d,temps)
>> xlabel('Day of month')
>> ylabel('Celsius')
>> title('Daily High Temperatures in three Cities')
```



Hình 13.1

Lệnh *plot* vừa dùng trên đây minh họa thêm một cách sử dụng. Biến *d* là một vector dài 31, trong khi biến *temps* là một ma trận 31x3. Cho trước những dữ liệu này, lệnh *plot* sẽ trích mỗi cột của biến *temps* cho vào *d*.

Để minh họa một vài khả năng phân tích dữ liệu của MATLAB, hãy xét các lệnh sau, dựa trên dữ liệu về nhiệt độ đã cho:

```
>> avg_temp = mean(temps)

avg_temp=

    11.9677    8.2258    19.8710
```

Vì dụ trên chỉ ra rằng thành phố thứ 3 là có nhiệt độ trung bình cao nhất, ở đây MATLAB đã tính nhiệt độ trung bình của mỗi cột một cách riêng rẽ. Nếu tính trung bình ở cả 3 thành phố thì:

```
>> avg_avg = mean(avg_temp)

avg_avg=

    13.3548
```

Khi mà các giá trị đầu vào trong một hàm phân tích dữ liệu là một vector hàng hay cột thì MATLAB chỉ đơn giản là tiến hành các phép toán trên vector và trả về giá trị số.

Bạn cũng có thể dùng mảng để thực hiện công việc này:

```
>> avg_temp = mean(temps,1) % Giống như trên, tính cho các cột

avg_temp =
```

```

11.9677  8.2258  19.8710
>> avr_tempr = mean(temps,2) % Tính cho mỗi hàng
avr_tempr =
12.6667
15.3333
12.0000
15.0000
13.3333
13.0000
13.0000
12.6667
14.6667
12.3333
14.3333
12.0000
13.0000
11.6667
13.6667
12.3333
11.3333
13.6667
12.0000
13.6667
13.3333
12.0000
14.3333
13.0000
12.6667

```

```
14.0000
13.0000
12.6667
16.0000
16.0000
15.3333
```

Đây là giá trị nhiệt độ trung bình ở cả ba thành phố trong từng ngày.

Xét bài toán tìm sự chênh lệch nhiệt độ của mỗi thành phố so với giá trị trung bình, có nghĩa là `avg_temp(i)` phải bị trừ đi bởi cột thứ *i* của biến `temps`. Bạn không thể ra một câu lệnh như sau:

```
>> temps-avg_temp
??? Error using ==> -
Matrix dimensions must agree.
```

Bởi vì thao tác này không phải là các thao tác đã định nghĩa trên mảng (`temps` là một mảng 31×3 , còn `avg_temp` là một mảng 1×3). Có lẽ cách dùng vòng lặp *for* là đơn giản nhất:

```
>> for i = 1:3
    tdev(:,i) = temps(:,i)- avg_temp(i);
end
>> tdev
tdev =
    0.0323 -0.2258 -1.8710
    3.0323  0.7742  2.1290
    0.0323 -3.2258 -0.8710
    2.0323 -0.2258  3.1290
    0.0323 -2.2258  2.1290
   -0.9677  0.7742 -0.8710
    3.0323  0.7742 -4.8710
   -3.9677  1.7742  0.1290
    7.0323 -1.2258 -1.8710
```

```

0.0323 -1.2258 -1.8710
2.0323 1.7742 -0.8710
-0.9677 -0.2258 -2.8710
-2.9677 -1.2258 3.1290
-3.9677 -0.2258 -0.8710
3.0323 -0.2258 -1.8710
-3.9677 0.7742 0.1290
-1.9677 -1.2258 -2.8710
0.0323 -1.2258 2.1290
-2.9677 -0.2258 -0.8710
0.0323 -0.2258 1.1290
0.0323 -0.2258 0.1290
-1.9677 0.7742 -2.8710
1.0323 3.7742 -1.8710
-2.9677 1.7742 0.1290
-1.9677 -2.2258 2.1290
2.0323 -1.2258 1.1290
0.0323 -3.2258 2.1290
1.0323 -1.2258 -1.8710
3.0323 1.7742 3.1290
1.0323 2.7742 4.1290
0.0323 3.7742 2.1290

```

Khi thực hiện phương pháp này ta thấy nó chậm hơn so với các câu lệnh được MATLAB thiết kế riêng để dung cho mảng. Nếu ta nhân bản biến `avg_temp` để kích thước của nó bằng với kích thước của `temps`, sau đó thực hiện phép trừ thì sẽ nhanh hơn rất nhiều:

```
>> tdev = temps - avg_temp(ones(31,1),:);
```

```
tdev =
```

```
0.0323 -0.2258 -1.8710
```

3.0323	0.7742	2.1290
0.0323	-3.2258	-0.8710
2.0323	-0.2258	3.1290
0.0323	-2.2258	2.1290
-0.9677	0.7742	-0.8710
3.0323	0.7742	-4.8710
-3.9677	1.7742	0.1290
7.0323	-1.2258	-1.8710
0.0323	-1.2258	-1.8710
2.0323	1.7742	-0.8710
-0.9677	-0.2258	-2.8710
-2.9677	-1.2258	3.1290
-3.9677	-0.2258	-0.8710
3.0323	-0.2258	-1.8710
-3.9677	0.7742	0.1290
-1.9677	-1.2258	-2.8710
0.0323	-1.2258	2.1290
-2.9677	-0.2258	-0.8710
0.0323	-0.2258	1.1290
0.0323	-0.2258	0.1290
-1.9677	0.7742	-2.8710
1.0323	3.7742	-1.8710
-2.9677	1.7742	0.1290
-1.9677	-2.2258	2.1290
2.0323	-1.2258	1.1290
0.0323	-3.2258	2.1290
1.0323	-1.2258	-1.8710
3.0323	1.7742	3.1290


```
1.0323  2.7742  4.1290
```

```
0.0323  3.7742  2.1290
```

Ở đây `avg_temp(ones(31,1),:)` sẽ nhân bản hàng đầu tiên (và là hàng duy nhất) của biến `avg_temp` thành 31 bản, tạo lên một ma trận 31×3 . Trong đó cột thứ i chính là `avg_temp(i)`.

```
>> max_temp = max(temps)
```

```
max_temp=
```

```
19  12  24
```

Câu lệnh tìm ra nhiệt độ lớn nhất ở mỗi thành phố trong tháng đó.

```
>> [max_temp,x] = max(temps)
```

```
max_temp=
```

```
19  12  24
```

```
x=
```

```
9  23  30
```

Cho biết giá trị nhiệt độ lớn nhất ở mỗi thành phố và giá trị chỉ số hàng x , tại đó giá trị lớn nhất xuất hiện, trong ví dụ này x cho biết ngày nóng nhất trong tháng.

```
>> min_temp = min(temps)
```

```
min_temp=
```

```
8  5  15
```

Cho biết nhiệt độ thấp nhất ở mỗi thành phố.

```
>> [min_temp, n] = min(temps)
```

```
min_temp=
```

```
8  5  15
```

```
n=
```

```
8  3  7
```

Cho biết giá trị nhiệt độ thấp nhất ở mỗi thành phố và chỉ số hàng n , tại đó giá trị thấp nhất xảy ra. Trong ví dụ này, n chính là ngày lạnh nhất trong tháng.

```
>> s_dev = std(temps)
```

```
s_dev=
```

```
2.5098  1.7646  2.2322
```

Cho biết độ chênh lệch chuẩn của biến temps.

```
>> daily_change = uiff(temps)
```

```
daily_change =
```

```
3  1  4
```

```
-3 -4 -3
```

```
2  3  4
```

```
-2 -2 -1
```

```
-1  3 -3
```

```
4  0 -4
```

```
-7  1  5
```

```
11 -3 -2
```

```
-7  0  0
```

```
2  3  1
```

```
-3 -2 -2
```

```
-2 -1  6
```

```
-1  1 -4
```

```
7  0 -1
```

```
-7  1  2
```

```
2 -2 -3
```

```
2  0  5
```

```
-3  1 -3
```

```
3  0  2
```

```
0  0 -1
```

```
-2  1 -3
```

```
3  3  1
```

```
-4 -2  2
```

```
1 -4  2
```

```
4  1 -1
```

```

-2 -2 1
1 2 -4
2 3 5
-2 1 1
-1 1 -2

```

Cho biết sự khác nhau về nhiệt độ giữa các ngày liên tiếp chính là độ chênh lệch nhiệt độ của ngày hôm sau so với ngày hôm trước. Trong ví dụ này, hàng đầu tiên của `daily_change` là độ chênh lệch nhiệt độ giữa ngày đầu tiên và ngày thứ hai trong tháng.

Các hàm phân tích dữ liệu

Phân tích dữ liệu trong MATLAB được thực hiện thông qua các ma trận hướng cột, các biến khác nhau được lưu giữ trong các cột khác nhau và mỗi hàm thể hiện giá trị của biến ở một thời điểm quan sát nhất định. Các hàm thống kê của MATLAB gồm có:

Các hàm phân tích dữ liệu	
<code>cplxpair(x)</code>	Xếp xếp cặp phức liên hợp
<code>cross(x,y)</code>	Tích chéo vector
<code>cumprod(x)</code>	Tích tích lũy theo cột
<code>cumprod(x,n)</code>	Tích tích lũy theo chiều n
<code>cumsum(x)</code>	Tổng tích lũy theo cột
<code>cumsum(x,n)</code>	Tổng tích lũy theo chiều n
<code>cumtrapz(x,y)</code>	Tích chéo tích lũy
<code>cumtrapz(x,y,n)</code>	Tích chéo tích lũy theo chiều n
<code>del2(A)</code>	Toán tử rời rạc Laplacian 5 điểm
<code>diff(x)</code>	Tính độ chênh lệch giữa các phần tử
<code>diff(x,m)</code>	Tính số ra cấp m của các phần tử
<code>diff(x,m,n)</code>	Tính số ra cấp m của các phần tử theo chiều n
<code>dot(x,y)</code>	Tích vô hướng của hai vector
<code>gradient(Z,dx,dy)</code>	Gradient vi phân

histogram(x)	Biểu đồ hình cột
max(x), max(x,y)	Phần tử lớn nhất
max(x,n)	Phần tử lớn nhất theo chiều n
mean(x)	Giá trị trung bình của cột
mean(x,n)	Giá trị trung bình theo chiều n
median(x)	Giá trị của phần tử giữa của cột
median(x,n)	Giá trị của phần tử giữa theo chiều n
min(x), min(x,y)	Phần tử nhỏ nhất
min(x,n)	Phần tử nhỏ nhất theo chiều n
prod(x)	Tích các phần tử trong cột
prod(x,n)	Tích các phần tử theo chiều n
rand(x)	Số ngẫu nhiên phân bố đều
randn(x)	Số ngẫu nhiên phân bố bình thường
sort(x)	Xếp xếp các cột theo thứ tự tăng dần
sort(x,n)	Xếp xếp theo chiều n
sortrows(A)	Xếp xếp các hàng theo thứ tự tăng dần
std(x), std(0)	Độ lệch chuẩn của cột chuẩn hoá theo N-1
std(x,1)	Độ lệch chuẩn của cột chuẩn hoá theo N
std(x, flag, n)	Độ lệch chuẩn theo chiều n
subspace(A,B)	Góc giữa hai điểm
sum(x)	Tổng các phần tử trong mỗi cột
sum(x,n)	Tổng các phần tử theo chiều n
trapz(x,y)	Tích chéo của $y=f(x)$
trapz(x,y,n)	Tích chéo theo chiều n

CÁC PHÉP TÍNH ĐỐI VỚI ĐA THỨC

14.1 Các nghiệm của đa thức

Tìm nghiệm của đa thức là tìm giá trị để đa thức bằng không, một bài toán thường gặp trong thực tế. MATLAB có thể giải quyết những bài toán này và đồng thời cung cấp những công cụ để tính toán đa thức. Trong MATLAB một đa thức được biểu diễn bằng một vector hàng các hệ số với bậc giảm dần. Ví dụ đa thức

$$x^4 - 12x^3 + 25x + 116 \quad \text{được nhập vào như sau:}$$

```
>> p = [1 -12 0 25 116]
```

```
p=
```

```
1    -12    0    25    116
```

Nhớ rằng mục dành cho hệ số 0 cũng phải được gõ vào nếu không MATLAB sẽ hiểu được hệ số của biểu thức bậc mấy là không. Sử dụng dạng này thì nghiệm của một đa thức có thể tìm được bằng cách dùng hàm *roots*.

```
>> r = roots(p)
```

```
r=
```

```
11.7374
```

```
2.7028
```

```
-1.2251 + 1.4672i
```

```
-1.2251 - 1.4672i
```

Bởi vì trong MATLAB cả đa thức và các nghiệm của nó đều là vector nên MATLAB ngầm quy ước rằng đa thức là vector hàng, còn các nghiệm là các vector cột. Nếu biết trước nghiệm

của một đa thức thì ta dễ dàng biết được đa thức đó. Trong MATLAB lệnh *poly* sẽ thực hiện công việc này.

```
>> pp = poly(r)

pp=

    1   -12    1.7764e-14    25    116

>> pp(abs(pp)< 1e-12) = 0           % Gán những phần tử quá nhỏ bằng không

    1   -12     0    25    116
```

Bởi vì trong tính toán thường gặp những sai số nên đôi khi kết quả của lệnh *poly* cho ra các đa thức có các hệ số gần bằng không và các đa thức có phần ảo rất nhỏ như được chỉ ra ở trên, các giá trị bằng không có thể được làm tròn bằng các công cụ vẽ mảng. Tương tự như vậy, ta có thể làm tròn một số phức để trở thành một số thực bằng hàm *real*.

14.2 Nhân đa thức

Hàm *conv* thực hiện nhân hai đa thức (thực ra là ma trận), xét tích của hai đa thức sau:

$$a(x) = x^3 + 2x^2 + 3x + 4 \text{ và } b(x) = x^3 + 4x^2 + 9x + 16$$

```
>> a = [1 2 3 4]; b = [1 4 9 16];
```

```
>> c = conv(a,b)
```

```
c=
```

```
    1     6    20    50    75    84    64
```

Kết quả là $c(x) = x^6 + 6x^5 + 20x^4 + 50x^3 + 75x^2 + 84x + 64$

Khi ta nhân nhiều đa thức với nhau thì ta phải sử dụng lệnh *conv* nhiều lần.

14.3 Phép cộng đa thức

MATLAB không cung cấp các hàm trực tiếp thực hiện phép cộng hai đa thức, dùng phép cộng ma trận chỉ có tác dụng khi hai đa thức là hai vector có cùng kích thước. Ví dụ như cộng hai đa thức $a(x)$ và $b(x)$ ở trên:

```
>> d = a + b
```

```
d=
```

```
    2     6    12    20
```

Kết quả là $d(x)=2x^3+6x^2+12x+20$. Khi hai đa thức có bậc khác nhau thì đa thức có bậc thấp hơn phải được thêm vào các hệ số 0 để cho bậc của nó có cùng bậc với đa thức có bậc cao hơn. Xét phép cộng hai đa thức c và d ở trên:

```
>> e = c + [0 0 0 d]
```

```
e=
```

```
1    6    20    52    81    96    84
```

Kết quả là $e(x)=x^6+6x^5+20x^4+52x^3+81x^2+84$. Các giá trị 0 cần phải được thêm vào ở phía đầu của vector chứ không phải phía đuôi, bởi vì các hệ số đó phải tương ứng với các hệ số bậc cao của x.

Nếu bạn muốn, bạn có thể tạo một hàm M_file để thực hiện phép cộng đa thức tổng quát:

```
function p=polyadd(a,b)
%POLYADD Polynomial addition
%POLYADD(A,B) adds the polynomials A and B
if nargin<2
    error('Not enough input arguments')
end
a=a(:).'; %make sure inputs are row vectors
b=b(:).';
na=length(a); %find lengths of a and b
nb=length(b);
p=[zeros(1,nb-na) a]+[zeros(1,na-nb) b];
% pad with zeros as necessary
```

Bây giờ có thể minh họa cho việc dùng hàm *polyadd*, hãy xét ví dụ trước đây:

```
>> f = polyadd(c,d)
```

```
f=
```

```
1    6    20    52    81    96    84
```

Kết quả cũng giống như đa thức e ở trên. Tất nhiên *polyadd* cũng có thể dùng để thực hiện phép trừ.

```
>> g = polyadd(c,-d)
```

```
g=
```

```
1 6 20 48 69 72 44
```

14.4 Chia hai đa thức

Trong một số trường hợp ta phải chia đa thức này cho một đa thức khác, trong MATLAB công việc này được thực hiện bởi hàm **deconv**, sử dụng các đa thức b và c ở trên ta có:

```
>> [q,r] = deconv(c,b)
```

```
q=
```

```
1 2 3 4
```

```
r=
```

```
0 0 0 0 0 0 0
```

Kết quả này chỉ ra rằng c đem chia cho b thì được đa thức là q và đa thức dư là r trong trường hợp này đa thức dư là đa thức 0 bởi vì c là đa thức chia hết cho q (nhớ rằng trên đây ta đã nhận được đa thức c bằng cách đem nhân đa thức a với đa thức b)

14.5 Đạo hàm

Bởi vì dễ dàng tính được vi phân của một đa thức nên MATLAB đưa ra hàm **polyder** để tính vi phân đa thức:

```
>> h = polyder(g)
```

```
h=
```

```
6 30 80 144 138 72
```

14.6 Tính giá trị của một đa thức

Rõ ràng rằng bạn có thể cộng, trừ, nhân, chia, đạo hàm một đa thức bất kỳ dựa trên các hệ số của nó, bạn cũng có thể dễ dàng tính được giá trị các đa thức này. Trong MATLAB hàm **polyval** sẽ thực hiện công việc này:

```
>> x = linspace(-1,3);
```

Sẽ chọn 100 điểm dữ liệu giữa -1 và 3

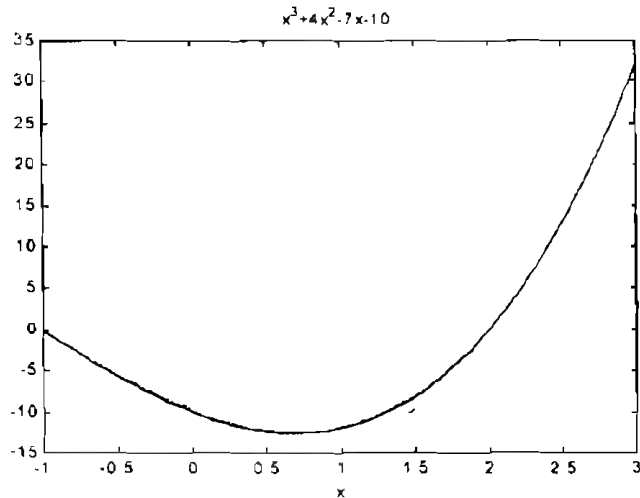
```
>> p = [1 4 -7 -10];
```


Dùng đa thức $p(x) = x^3 + 4x^2 - 7x - 10$

```
>> v = polyval(p,x);
```

Tính giá trị của $p(x)$ tại các giá trị của x và lưu trữ kết quả vào trong mảng v . Sau đó kết quả sẽ được vẽ ra bằng lệnh plot (hình 14.1):

```
>> plot(x, v), title('x^3+4x^2-7x-10'), xlabel('x')
```



Hình 14.1

14.7 Phân thức hữu tỉ

Đôi khi bạn gặp những bài toán liên quan đến tỉ số của hai đa thức hay còn gọi là phân thức hữu tỉ, ví dụ như các hàm truyền hay các hàm xấp xỉ *pade* có dạng như sau:

$$\frac{n(x)}{d(x)} = \frac{N_1 x^m + N_2 x^{m-1} + \dots + N_{m-1}}{D_1 x^n + D_2 x^{n-1} + \dots + D_{n+1}}$$

Trong MATLAB phân thức cũng được mô phỏng bằng hai đa thức riêng rẽ. Ví dụ như:

```
>> n=[1 -10 100] % a numerator
```

n =

```
1 -10 100
```

```
>> d=[1 10 100 0] % a denominator
```

d =

```
1 10 100 0
```

```
>> z=roots(n)    % the zeros of n(x)/d(x)
```

```
z =
```

```
5.0000 + 8.6603i
```

```
5.0000 - 8.6603i
```

```
>> p=roots(d)    % the poles of n(x)/d(x)
```

```
p =
```

```
0
```

```
-5.0000 + 8.6603i
```

```
-5.0000 - 8.6603i
```

Đạo hàm của phân thức này theo biến x được tính dựa trên hàm *polyder*.

```
>> [nd,dd]=polyder(n,d)
```

```
nd =
```

```
-1    20   -100  -2000 -10000
```

```
dd =
```

```
Columns 1 through 6
```

```
1    20    300    2000   10000    0
```

```
Column 7
```

```
0
```

ở đây nd và dd là tử thức và mẫu thức của đạo hàm. Một thao tác thông thường khác là tìm phần dư của phân thức.

```
>> [r,p,k]=residue(n,d)
```

```
r =
```

```
0.0000 + 1.1547i
```

```
0.0000 - 1.1547i
```

```
1.0000
```

```
p =
```

```
-5.0000 + 8.6603i
```

```
-5.0000 - 8.6603i
```

```
0
```

k -

[]

Trong trường hợp này hàm *residue* trả về các hệ số mở rộng phân thức từng phần r, các nghiệm của phân thức là p và phần thương chia hết của phân thức là k. Nếu bậc của tử số nhỏ hơn bậc của mẫu số thì phân thức chia hết sẽ bằng không. Trong ví dụ trên thì mở rộng phân thức từng phần của phân thức đã cho là:

$$\frac{n(x)}{d(x)} = \frac{1.1547i}{x+5-8.6603i} + \frac{-1.1547i}{x-5+8.6603i} + \frac{1}{x}$$

Nếu cho trước các đa thức này thì phân thức ban đầu sẽ tìm được bằng cách sử dụng hàm *residue*:

```
>> [nn,dd]=residue(r,p,k)
```

```
nn =
```

```
1.0000 -10.0000 100.0000
```

```
dd =
```

```
1.0000 10.0000 100.0000 0
```

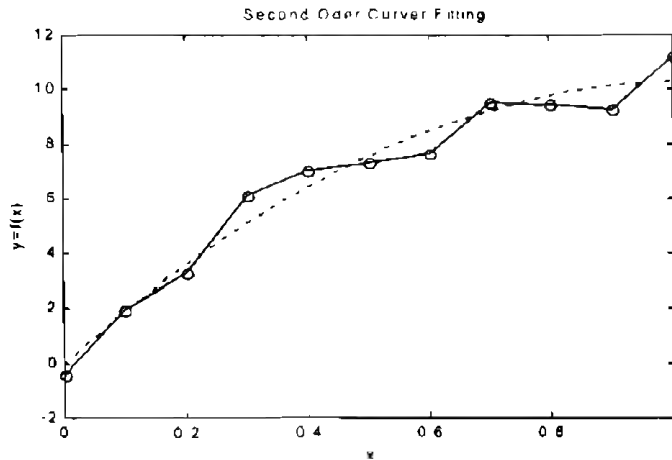
Vì vậy trong trường hợp này, hàm *residue* có thể thực hiện được việc chuyển đổi hai chiều tùy thuộc vào số lượng các tham số vào và ra truyền cho nó.

PHÉP NỘI SUY VÀ MỊN HOÁ ĐƯỜNG CONG

Trong các lĩnh vực ứng dụng số, nhiệm vụ của chúng ta là phải biểu diễn số liệu, thường là các số đo bằng các chức năng phân tích. Có hai cách giải quyết vấn đề này, trong phương pháp nội điểm (interpolation) thì dữ liệu được coi là đúng và cái chúng ta cần là cách biểu diễn dữ liệu không nằm giữa các giá trị đo được, theo phương pháp thứ hai gọi là phương pháp mịn hoá đường cong (curve fitting or regression), bạn tìm một đường cong không gây khúc mà phù hợp nhất với dữ liệu đã có, nhưng không cần thiết phải đi qua một cách chính xác bất kỳ một điểm nào trên bảng số liệu. Hình 15.1 minh họa hai phương pháp trên, chữ o đánh dấu các điểm biểu diễn dữ liệu, các đoạn thẳng bằng nét liền nối các đường biểu diễn dữ liệu lại với nhau theo phương pháp nội điểm còn đường chấm chấm là một đường cong vẽ theo phương pháp mịn hoá dữ liệu.

15.1 Mịn hoá đường cong

Phương pháp mịn hoá đường cong liên quan đến việc trả lời hai câu hỏi cơ bản, đó là đường cong thế nào thì phù hợp với dữ liệu nhất và câu hỏi thứ hai là phải sử dụng loại đường cong nào. "Phù hợp nhất" có thể hiểu theo nhiều cách và do đó có nhiều đường cong, vì vậy chúng ta phải bắt đầu từ đâu? Nếu "phù hợp nhất" là giảm nhỏ đến mức tối thiểu tổng sai số quân phương tại mỗi điểm biểu diễn dữ liệu, so với giá trị tương ứng trên đường cong thì đường cong phù hợp nhất sẽ là một đường thẳng về mặt toán mà nói phương pháp này được gọi là phương pháp xấp xỉ đa thức. Nếu như khái niệm này còn khó hiểu đối với bạn thì xin hãy xem lại hình 15.1 khoảng cách theo chiều dọc giữa đường cong dữ liệu và các điểm biểu diễn dữ liệu gọi là sai số của điểm đó, bình phương khoảng cách này lên và cộng tất cả chúng lại ta được tổng bình phương sai số. Đường cong chấm chấm là đường cong làm cho bình phương sai số là nhỏ nhất và được gọi là đường cong phù hợp nhất. Từ "quân phương bé nhất" là cách nói tắt của cụm từ "Tổng bình phương sai số bé nhất".



Hình 15.1

Trong MATLAB hàm *polyfit* sẽ giải quyết vấn đề xấp xỉ đường cong qua phương bé nhất. Để minh họa cho việc sử dụng hàm này, chúng ta hãy bắt đầu bằng các dữ liệu đã có ở trong hình vẽ.

```
>> x = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1];
```

```
>> y = [-.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.30 11.2];
```

Để sử dụng hàm *polyfit*, chúng ta phải truyền cho nó dữ liệu trên và bậc của đa thức mà chúng ta muốn phù hợp với dữ liệu, nếu chúng ta chọn bậc n là 1 thì đường cong xấp xỉ gần nhất sẽ là đường thẳng. Phương pháp này được gọi là phương pháp xấp xỉ tuyến tính. Mặt khác nếu chúng ta chọn $n=2$ thì chúng ta sẽ tìm được một tam thức bậc hai. Ví dụ:

```
>> n = 2;
```

```
>> p = polyfit(x,y,n)
```

```
p =
```

```
-9.8108 20.1293 -0.0317
```

Kết quả của *polyfit* là một vector biểu diễn hệ số của một đa thức bậc hai. Ở đây đa thức đó là

$y = -9.8108x^2 + 20.1293x - 0.0317$. Để so sánh mức độ xấp xỉ của đa thức với các điểm dữ liệu chúng ta hãy vẽ hai đường:

```
>> xi = linspace(0,1,100);
```

Dòng này để tạo ra dữ liệu trục x để chuẩn bị vẽ đa thức.

```
>> z = polyval(p,xi)
```

Dòng này gọi hàm ***polyval*** của MATLAB để tính giá trị của đa thức p tại các điểm xi

```
>> plot(x,y,'-o',xi,z,':')
```

Vẽ các điểm có tọa độ là x và y, đánh dấu các điểm này bằng chữ 'o' sau đó nối các điểm này bằng các đoạn thẳng. Ngoài ra nó còn vẽ dữ liệu của đa thức xi và z dùng đường chấm chấm.

```
>> xlabel('x'),ylabel('y=f(x)')
```

```
>> title('Second Oder Curver Fitting')
```

Tạo nhãn cho đường cong vừa vẽ. Kết quả của các lệnh trên đây là một đồ thị đã được giới thiệu ở trên.

Việc chọn bậc của đa thức không phải là ngẫu nhiên, nếu có hai điểm thì xác định một đường thẳng, tức là một đa thức bậc nhất, ba điểm thì xác định một parabol bậc hai. Cứ như vậy, để xác định một đường cong bậc n, cần có n+1 điểm. Vì vậy, ở trong ví dụ trước có 11 điểm dữ liệu, chúng ta có thể chọn bậc của đa thức là từ 1 đến 10. Tuy nhiên, do tính chất số học của các đa thức bậc cao rất phức tạp nên bạn không nên chọn bậc của đa thức lớn hơn mức cần thiết. Ngoài ra khi bậc của đa thức tăng lên thì sự xấp xỉ càng kém hơn, vì vậy các đa thức bậc cao có thể bị đạo hàm nhiều lần trước khi đạo hàm của chúng bằng không. Ví dụ cho một đa thức bậc 10:

```
>> pp = polyfit(x,y,10)
```

```
pp =
```

```
1.0e+006 *
```

```
Columns 1 through 7
```

```
-0.4644  2.2965 -4.8773  5.8233 -4.2948  2.0211 -0.6032
```

```
Columns 8 through 11
```

```
0.1090 -0.0106  0.0004 -0.0000
```

```
>> format short e    % change display format
```

```
>> pp.' % display polynomial coefficients as a column
```

```
ans =
```

```
-4.6436e+005
```

```
2.2965e+006
```

```
-4.8773e+006
```

5.8233e+006

-4.2948e+006

2.0211e+006

-6.0322e+005

1.0896e+005

-1.0626e+004

4.3599e+002

-4.4700e-001

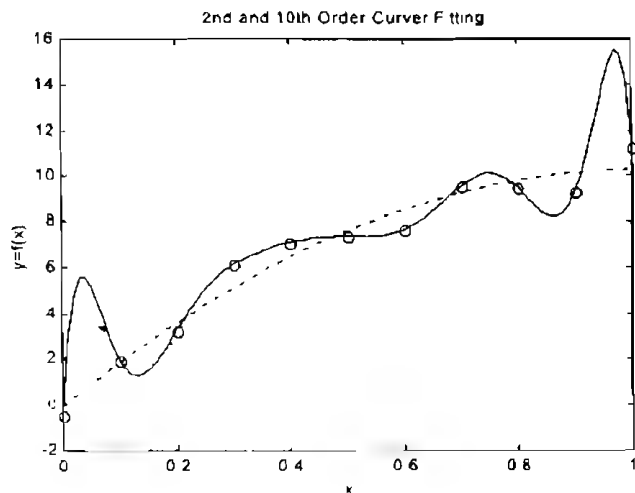
Lưu ý kích thước của vector hệ số đa thức trong trường hợp này so với đường cong bậc hai trước đây, đồng thời cũng lưu ý sự khác nhau giữa số hạng nhỏ nhất và số hạng lớn nhất trong đa thức vào khoảng 10^7 . Hãy thử vẽ đường cong này và so sánh với dữ liệu gốc và với đường cong bậc hai.

```
>> zz = polyval(pp,xi); % evaluate 10th order polynomial
```

```
>> plot(x,y,'o',xi,z,'-',xi,zz) % plot data
```

```
>> xlabel('x'),ylabel('y=f(x)')
```

```
>> title('2nd and 10th Order Curver F tting')
```



Hình 15.2

Trên hình 15.2, dữ liệu gốc được đánh dấu o, đường cong bậc hai được vẽ bằng nét chấm chấm, còn đường cong bậc 10 được vẽ bằng nét đậm. Để ý đến nét gợn sóng xuất hiện giữa các

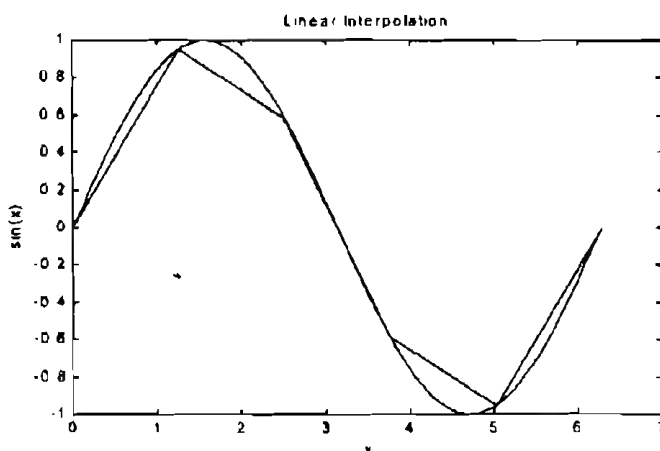
điểm dữ liệu bên phía trái và bên phía phải của đường cong bậc 10. Dựa vào đồ thị này thì rõ ràng rằng cái triết lý càng nhiều càng tốt không thể áp dụng được ở đây.

15.2 Nối điểm một chiều

Như đã giới thiệu thì nối điểm được định nghĩa như là một phương pháp dự đoán giá trị của hàm giữa những điểm cho trước. Nối điểm là một công cụ hữu hiệu khi chúng ta không thể nhanh chóng tính được giá trị của hàm tại các điểm trung gian. Phương pháp này được sử dụng rộng rãi đối với dữ liệu là giá trị của các phép đo thực nghiệm hoặc là kết quả của các chuỗi tính toán dài. Có thể ví dụ đơn giản nhất của việc nối điểm chính là phương pháp vẽ từng điểm của MATLAB, tức là vẽ những đoạn thẳng nối những điểm dữ liệu liên tiếp để tạo lên một đồ thị.

Đây là phương pháp nối điểm tuyến tính, nó cho rằng các giá trị của hàm nằm giữa hai điểm cho trước sẽ rơi vào khoảng giữa hai đầu của đoạn thẳng nối hai điểm đó. Hiển nhiên là khi số lượng các điểm dữ liệu tăng lên và khoảng cách giữa chúng giảm đi thì phương pháp nối điểm tuyến tính càng trở lên chính xác.

```
>> x1 = linspace(0,2*pi,60);  
>> x2 = linspace(0,2*pi,6);  
>> plot(x1,sin(x1),x2,sin(x2),'-')  
>> xlabel('x'),ylabel('sin(x)')  
>> title('Linear Interpolation')
```



Hình 15.3

Cả hai đồ thị cùng vẽ một hàm sine nhưng đồ thị 60 điểm thì mịn hơn đồ thị 6 điểm

Cũng giống như phương pháp xấp xỉ hoá đường cong, ở đây chúng ta cũng phải thực hiện một số lựa chọn, có rất nhiều cách để nối hai điểm, tùy thuộc vào giả định mà chúng ta đã lựa chọn. Hơn nữa chúng ta có thể nối các điểm trong không gian không phải là một chiều. Nói như thế nếu bạn có dữ liệu phản ánh một hàm phụ thuộc vào hai biến $z=f(x,y)$, bạn có thể nối giá trị nằm giữa hai điểm có x và y khác nhau để tìm ra giá trị trung gian của hai điểm. MATLAB cung cấp một số hàm để nối là: *interp1* nối các dữ liệu một chiều, *interp2* nối các dữ liệu hai chiều, *interp3* nối các dữ liệu ba chiều, *interp* nối các dữ liệu có số chiều lớn hơn 3

Sau đây chúng ta sẽ xem xét các dữ liệu một và hai chiều. Để minh họa việc nối dữ liệu một chiều, hãy xét ví dụ sau, khả năng của thính giác, ví dụ như mức âm thanh bé nhất nay còn gọi là ngưỡng nghe của tai người thay đổi theo tần số, dữ liệu do người thống kê được cho như sau:

```
>> Hz = [20:10:100 200:100:1000 1500 2000:1000:10000];
>> % Frequencies in Hertz
>> spl = [76 66 59 54 49 46 43 40 38 22 ...
14 9 6 3.5 2.5 1.4 0.7 0 -1 -3 ...
-8 -7 -2 2 7 9 11 12];
>> % sound pressure level in dB
```

Ngưỡng nghe được chuẩn hoá bằng 0dB tại tần số 1000Hz, bởi vì tần số trải trong một dải rất rộng nên khi vẽ các điểm dữ liệu chúng ta logarithm hoá trục x.

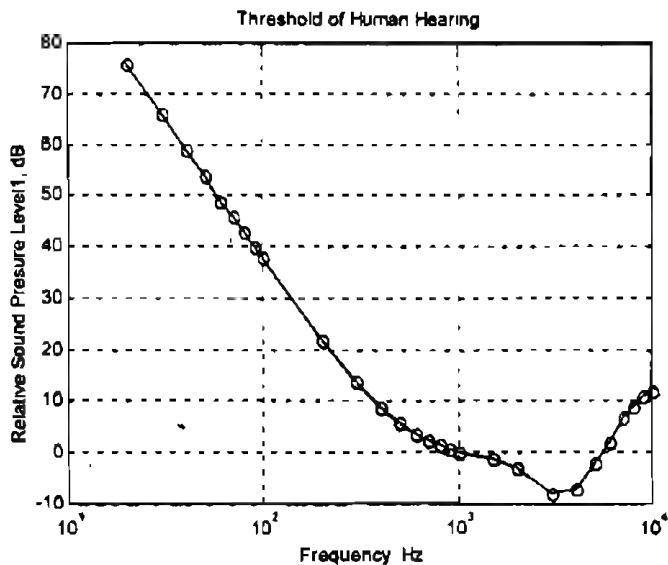
```
>> semilogx(Hz,spl,'o')
>> xlabel('Frequency, Hz')
>> ylabel('Relative Sound Pressure Level1, dB')
>> title('Threshold of Human Hearing')
```

Dựa vào hình 15.4 ta thấy tai người nhạy cảm hết đối với các âm thanh trong khoảng 3kHz. Dựa vào các số liệu này, chúng ta hãy dự đoán ngưỡng nghe ở tần số 2,5kHz bằng một vài cách khác nhau

```
>> s = interp1(Hz,spl,2.5e3) %linear interpolation
s =
-5.5000e+000
>> s = interp1(Hz,spl,2.5e3,'linear') %linear interpolation again
s =
-5.5000e+000
```

```
>> s = interp1(Hz,spl,2.5e3,'cubic') % cubic interpolation
s =
-5.8690e+000
>> s = interp1(Hz,spl,2.5e3,'spline') % spline interpolation
s =
-5.8690e+000
>> s = interp1(Hz,spl,2.5e3,'nearest')% nearest-neighbor
s =
-8
```

Hãy để ý đến sự khác nhau trong các kết quả, hai giá trị đầu tiên trả về một cách chính xác giá trị được vẽ ở trên hình 15.4 tại tần số 2,5 kHz bởi vì MATLAB đã nối các điểm một cách tuyến tính giữa các điểm dữ liệu trên đồ thị các đường cong đa thức, ví dụ như đa thức bậc 3 sẽ xấp xỉ hoá các điểm trên đồ thị theo các cách khác nhau, kết quả là các đường cong này tương đối phù hợp với các dữ liệu mà nó đi qua trên đồ thị nhưng khác biệt khá xa so với phương pháp nối bằng đường thẳng.



Hình 15.4

Vì vậy bạn chọn cách nào để giải quyết một bài toán cho trước?, trong nhiều trường hợp thì chỉ cần nối một cách tuyến tính là đủ, trong thực tế thì đó chính là phương pháp mặc định khi các

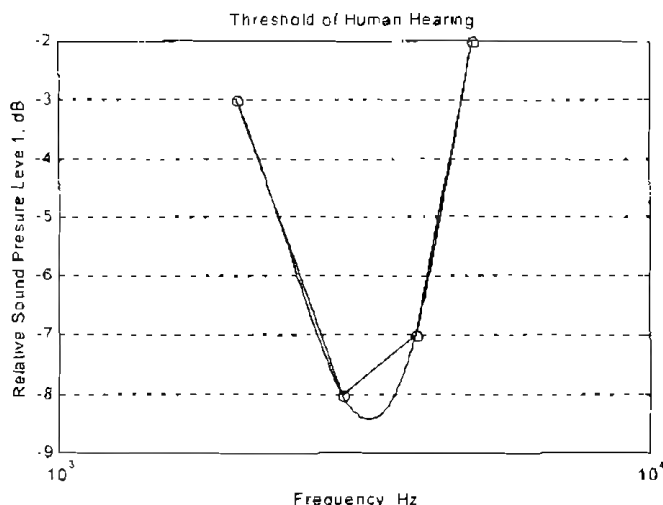
đường cong càng gần với các đoạn thẳng thì càng kém chính xác nhưng ngược lại tốc độ tính toán nhanh, điều này đặc biệt quan trọng khi tập dữ liệu lớn. Một phương pháp tiêu tốn nhiều thời gian, cho ra kết quả đẹp mắt nhưng không hiệu quả. Trong thực tế một trong những tác dụng chủ yếu của phương pháp nối điểm bằng hàm bậc 3 hoặc cao hơn là để mịn hoá dữ liệu, có nghĩa là cho trước một tập dữ liệu ta có thể dùng phương pháp này để tính ra giá trị của hàm ở những thời điểm nhất định bất kỳ. Ví dụ:

```
>> Hzi = linspace(2e3,5e3);      % look closely near minimum
>> spli = interp1(Hz,spl,Hzi,'cubic'); % interpolate near minimum
>> i = find(Hz>=2e3&Hz<=5e3);
>> % find original data indices near minimum
>> semilogx(Hz(i),spl(i),'-o',Hzi,spli) % plot old and new data
>> xlabel('Frequency, Hz')
>> ylabel('Relative Sound Pressure Level1, dB')
>> title('Threshold of Human Hearing')
>> grid on
```

Hình 15.5 đường nét rời sử dụng phương pháp nối điểm tuyến tính, đường liền nét là một hàm bậc 3, còn những điểm dữ liệu gốc được đánh dấu bởi chữ o. Bằng cách nâng cao độ phân giải trên trục tần số và sử dụng đường bậc 3 thì các số liệu về ngưỡng nghe mà chúng ta dự đoán được sẽ mịn hơn.

Cần chú ý rằng độ dốc của đường bậc 3 không thay đổi một cách đột ngột khi đi qua điểm dữ liệu như là khi sử dụng phương pháp nối tuyến tính. Với bộ dữ liệu trên chúng ta có thể dự đoán được tần số mà tại đó tai người nhạy cảm nhất đối với âm thanh.

```
>> [sp_min,i] = min(spli) % minimum and index of minimum
sp_min =
-8.4245e+000
i =
45
>> Hz_min = Hzi(i) % frequency at minimum
Hz_min =
3.3333e+003
```



Hình 15.5

Tại người nhạy cảm nhất đối với âm thanh có tần số khoảng 3.3 kHz. Trước khi đề cập đến việc xấp xỉ hoá hai chiều thì chúng ta cần nhận rõ hai hạn chế lớn của *interp1* là: Thứ nhất khi yêu cầu tính toán ở ngoài khoảng của một biến độc lập. Ví dụ như *interp1*(Hz, spl, 1e5) thì sẽ sinh ra kết quả NaN.

Thứ hai là các biến độc lập phải đơn điệu, nghĩa là các biến độc lập phải luôn tăng hoặc là luôn giảm. Trong ví dụ trên của chúng ta thì trục tần số Hz luôn tăng.

15.3 Xấp xỉ hoá hai chiều

Xấp xỉ hoá hai chiều dựa trên cùng một nguyên lý của xấp xỉ hoá một chiều. Tuy nhiên như tên của nó đã chỉ ra, xấp xỉ hoá hai chiều là xấp xỉ một hàm phụ thuộc vào hai biến độc lập:

$$z = f(x, y)$$

Để hiểu rõ khái niệm này, ta hãy xét ví dụ sau:

Một công ty thám hiểm đại dương, cần thám hiểm một vùng biển, cứ 0,5 km theo hình vuông thì độ sâu của đáy biển lại được đo và ghi lại một phần của dữ liệu thu thập được vào trong một chương trình MATLAB dưới dạng một M-file có tên là ocean.m như sau:

```
function ocean
% ocean depth data
x=0:0.5:4; % x-axis (varies across the rows of z)
y=0:0.5:6; % y-axis (varies down the columns of z)
z=[100 99 100 99 100 99 99 99 100
```

```

100 99 99 99 100 99 100 99 99
99 99 98 98 100 99 100 100 100
100 98 97 97 99 100 100 100 99
101 100 98 98 100 102 103 100 100
102 103 101 100 102 106 104 101 100
99 102 100 100 103 108 106 101 99
97 99 100 100 102 105 103 101 100
100 102 103 101 102 103 102 100 99
100 102 103 102 101 101 100 99 99
100 100 101 101 100 100 100 99 99
100 100 100 100 100 99 99 99 99
100 100 100 99 99 100 99 100 99];

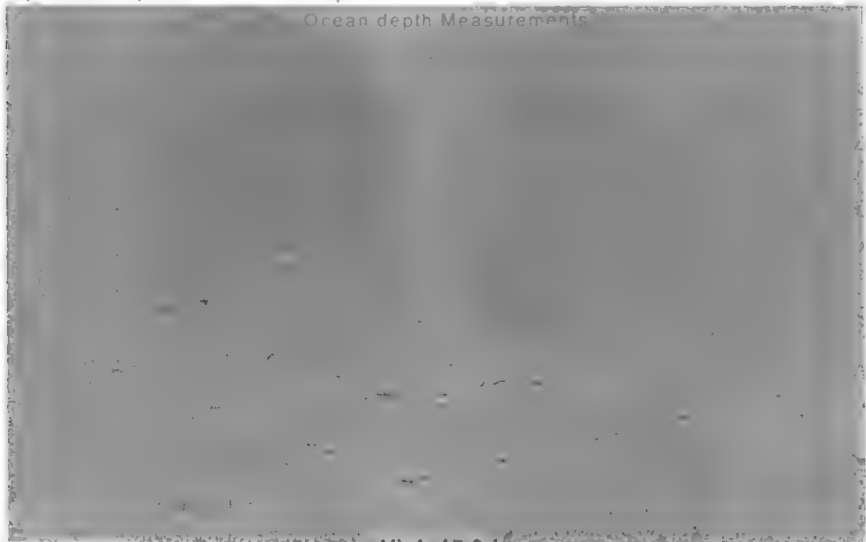
```

Đồ thị của dữ liệu trên được vẽ bởi các lệnh sau (hình 15.6)

```

mesh(x,y,z)
xlabel('X-axis, Km')
ylabel('Y-axis, Km')
zlabel('Ocean depth, m')
title('Ocean depth Measurements')

```



Hình 15.6

Sử dụng các dữ liệu này thì độ sâu của một điểm bất kỳ nằm trong khu vực khảo sát có thể tính được dựa vào hàm *interp2*. Ví dụ:

```
>> zi = interp2(x,y,z,2.2,3.3)

zi =

    1.0392e+002

>> zi = interp2(x,y,z,2.2,3.3,'linear')

zi =

    1.0392e+002

>> zi = interp2(x,y,z,2.2,3.3,'cubic')

zi =

    1.0419e+002

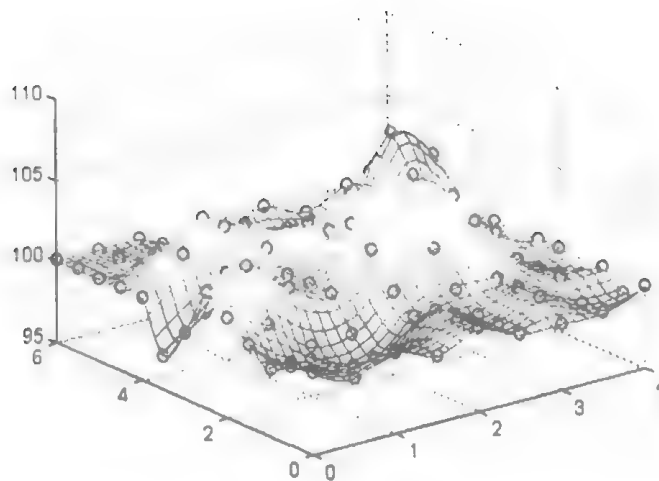
>> zi = interp2(x,y,z,2.2,3.3,'nearest')

zi =

    102
```

Cũng giống như trong trường hợp xấp xỉ hoá một chiều, xấp xỉ hoá hai chiều cũng có nhiều phương pháp, mà phương pháp đơn giản nhất là phương pháp nối bằng đoạn thẳng, hay còn gọi là nối tuyến tính. Một lần nữa chúng ta có thể xấp xỉ hoá để cho đồ thị trở lên mịn hơn với độ phân giải cao hơn:

```
xi=linspace(0,4,30); % finer x-axis
yi=linspace(0,6,40); % finer y-axis
[xxi,yyi]=meshgrid(xi,yi);
% grid of all combinations of xi and yi
zzi=interp2(x,y,z,xxi,yyi,'cubic'); % interpolate
mesh(xxi,yyi,zzi) % smoothed data
hold on
[xx,yy]=meshgrid(x,y); % grid original data
plot3(xx,yy,z+0.1,'ok')
% plot original data up a bit to show nodes
hold off
```



Hình 15.7

Ở đây hàm **meshgrid** được dùng để tạo mảng xấp xỉ hoá bao phủ toàn bộ những điểm yêu cầu nằm trong điểm khảo sát. Như trong hình 15.7, hàm **meshgrid** thực hiện điều đó bằng cách tạo ra một mảng hai chiều dựa trên các vector x_i và y_i , sử dụng mảng này chúng ta có thể dự đoán được chỗ nông nhất của đáy biển.

```
>> zmax = max(max(zzi))
```

```
zmax=
```

```
108.05
```

```
>> [i,j] = find(zmax==zzi);
```

```
>> xmax = xi(j)
```

```
xmax=
```

```
2.6207
```

```
>> ymax = yi(j)
```

```
ymax=
```

```
2.9231
```

PHÂN TÍCH SỐ LIỆU

Việc giải một bài toán tích phân hoặc tính giá trị của một hàm là tương đối phức tạp, nhưng đối với máy tính thì đó chỉ là một việc đơn giản. Lĩnh vực này của tin học và toán học được gọi là xử lý số liệu. Như bạn có thể dự đoán, MATLAB cung cấp các công cụ để giải quyết vấn đề này. Trong chương này chúng ta xem xét cách sử dụng các công cụ đó.

16.1 Vẽ đồ thị

Cho đến thời điểm này thì việc vẽ đồ thị của một hàm vẫn chỉ đơn giản dựa trên việc tính giá trị của hàm đó tại một số điểm rời rạc, và dùng các điểm để biểu diễn các hàm tại các giá trị rời rạc đó. Trong nhiều trường hợp thì giải pháp này là có thể chấp nhận được. Tuy nhiên có một số hàm thì tương đối bằng phẳng ở một số khoảng nào đó nhưng lại trở lên đột biến ở một số giá trị nhất định. Sử dụng phương pháp vẽ truyền thống trong trường hợp này có thể làm mất đi tính chân thực của đồ thị. Vì vậy MATLAB cung cấp cho ta một hàm vẽ đồ thị thông minh, gọi là *fplot*. Hàm này tính toán một cách cẩn thận hàm số cần vẽ và đảm bảo một cách chắc chắn rằng tất cả các điểm đặc biệt được biểu diễn trên đồ thị. Hàm *fplot* nhận vào là tên của hàm cần vẽ dưới dạng một chuỗi kí tự, và giá trị cần vẽ dưới dạng mảng gồm hai phần tử chứa giá trị đầu và giá trị cuối. Ví dụ:

```
>> fplot('humps',[0 2])
>> title('Fplot of humps')
```

Tính các giá trị của hàm *humps* nằm giữa 0 và 2 và thể hiện đồ thị trong hình 16.1. Trong ví dụ này *humps* là một hàm M-file thiết kế sẵn.

```
function [out1,out2] = humps(x)

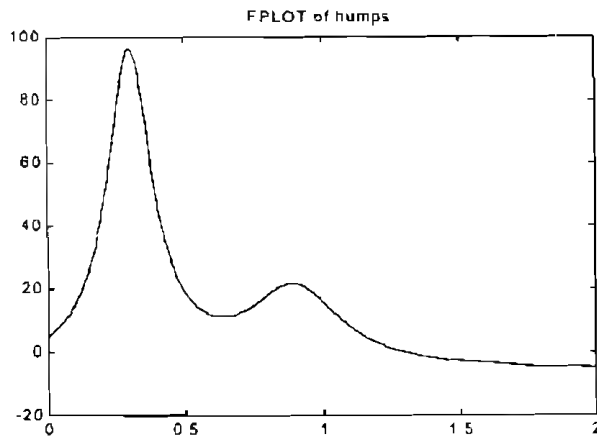
%HUMPS A function used by QUADDEMO, ZERODEMO and FPLOTEDEMO.

% Y = HUMPS(X) is a function with strong maxima near x = .3

% and x = .9.
```



```
%
% [X,Y] = HUMPS(X) also returns X. With no input arguments,
% HUMPS uses X = 0:.05:1.
%
```



Hình 16.1

```
% Example:
% plot(humps)
%
% See QUADDEMO, ZERODEMO and FPLOTDEMO.
% Copyright (c) 1984-98 by The MathWorks, Inc.
% $Revision: 5.4 $ $Date: 1997/11/21 23:26:10 $
if nargin==0, x = 0:.05:1; end
y = 1 ./ ((x-3).^2 + .01) + 1 ./ ((x-.9).^2 + .04) - 6;
if nargout==2,
    out1 = x; out2 = y;
else
    out1 = y;
end
```

Hàm *fplot* làm việc với bất cứ một hàm M_file nào có một giá trị vào và một giá trị ra, nghĩa là giống như hàm *humps* ở trên, biến ra y trả về một mảng có cùng kích thước với biến vào x. Một lỗi thường xảy ra khi sử dụng hàm *fplot* cũng giống như khi sử dụng các hàm phân tích số khác là bỏ quên dấu nhảy đơn ở tên hàm cần vẽ. Hàm *fplot* cần dấu nhảy đơn đó để tránh nhầm lẫn tên hàm với các biến trong môi trường MATLAB. Đối với các hàm đơn giản được biểu diễn bằng một chuỗi các kí tự. Ví dụ $y = 2.e^{\sin(x)}$ thì hàm *fplot* có thể vẽ được đồ thị của hàm trên mà không cần phải tạo ra một M_file. Để thực hiện điều đó chỉ cần viết hàm cần vẽ dưới dạng một chuỗi kí tự có sử dụng x là biến số độc lập.

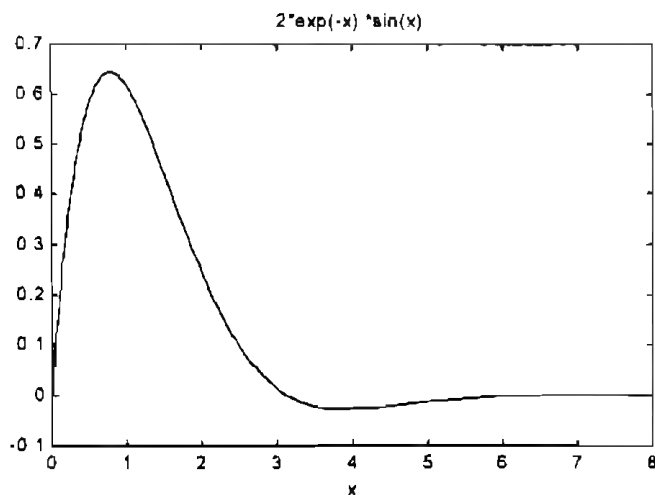
```
>> f = '2*exp(-x).*sin(x)';
```

Ở đây hàm $f(x) = 2.e^{\sin(x)}$ được định nghĩa bằng cách sử dụng phép nhân ma trận.

```
>> fplot(f,[0 8])
```

```
>> title(f), xlabel('x')
```

Vẽ đồ thị của hàm nằm trong khoảng từ 0 đến 8 tạo ra đồ thị như hình 16.2.



Hình 16.2

Dựa trên những tính năng cơ bản này, hàm *fplot* có những khả năng rất mạnh, hãy xem phần trợ giúp trực tuyến của MATLAB để hiểu rõ hơn về cách dùng hàm này.

16.2 Cực trị của một hàm

Ngoài việc sử dụng phương pháp vẽ đồ thị để thu được những thông tin trực quan về hàm, chúng ta còn cần phải biết thêm những thông tin về một số thuộc tính nhất định của hàm. Trong nhiều trường hợp chúng ta cần phải biết các cực trị của hàm đó, đó là các cực đại, các cực tiểu. Về mặt toán học thì cực trị được tìm theo phương pháp giải tích bằng cách tính đạo hàm của

hàm đó và tìm những điểm mà tại đó đạo hàm bằng 0. Điều này rất dễ hiểu nếu bạn xem lại đồ thị của hàm *humps* nói trên. Những điểm mà đồ thị của hàm nhô lên tạo ra những điểm cực đại, còn những điểm đồ thị lõm xuống thấp nhất là những điểm cực tiểu. Rõ ràng rằng khi hàm được định nghĩa một cách đơn giản thì phương pháp giải tích có thể dễ dàng thực hiện được, tuy nhiên đối với một số hàm cho dù việc tính đạo hàm là khá dễ dàng thì việc tìm nghiệm của đạo hàm thì lại không phải là đơn giản. Trong những trường hợp này, và trong những trường hợp khó có thể tìm ra cách phân tích đạo hàm, thì cần thiết phải tìm hàm vô cùng về số lượng. MATLAB cung cấp hai hàm thực hiện việc này, đó là *fmin* và *fmins* hai hàm này tương ứng tìm giá trị cực tiểu của các hàm một chiều và hàm n chiều. Ta chỉ quan tâm đến *fmin* trong phần này. Hơn nữa *fmin* có thể tìm thấy trong help trực tuyến. Bởi vì max của $f(x)$ hoàn toàn tương đương với min của $-f(x)$, nên *fmin* và *fmins*, cả hai đều được dùng để tìm giá trị lớn nhất và nhỏ nhất.

Để minh họa phép cực tiểu hoá và cực đại hoá, hãy xem ví dụ trước đó một lần nữa. Từ hình 16.2 có một giá trị cực đại gần $x_{\max}=0.7$ và một giá trị nhỏ nhất gần $x_{\min}=4$. Điều này có thể cho phép ta xem như $x_{\max}=\pi/4\approx 0.785$, $x_{\min}=5\pi/4\approx 3.93$. Viết ra một script-file dùng chế độ soạn thảo thuận tiện và sử dụng *fmin* để tìm ra số này:

```
function ex_fmin.m
%ex_fmin.m

fn='2*exp(-x)*sin(x)'; % define function for min
xmin=fmin(fn,2,5) % search over range 2<x<5
emin=5*pi/4-xmin % find error
x=xmin; % eval needs x since fn has x
% as its variable
ymin=eval(fn) % evaluate at xmin
fx='-2*exp(-x)*sin(x)'; % define function for max:
% note minus sign
xmax=fmin(fn,0,3) % search over range 0<x<3
emax=pi/4-xmax % find error
x=xmax; % eval needs x since fn has x
% as its variable
ymax=eval(fn) % evaluate at xmax
```

Chạy M_file này thì kết quả như sau:

```
xmin =
```

```

3.9270
emin =
1.4523e-006
ymin =
-0.0279
xmax =
3.0000
emax =
-2.2146
ymax =
0.0141

```

Kết quả này hoàn toàn phù hợp với đồ thị trước đó. Chú ý rằng *fmin* làm việc nói chung là như *fplot*. Ví dụ này còn giới thiệu hàm *eval*, hàm này nhận một chuỗi ký tự và giải thích nó như là chuỗi được đánh vào từ dấu nhắc của MATLAB.

Cuối cùng, một điều quan trọng cần chú ý khác là việc tối thiểu hoá liên quan đến việc tìm giá trị nhỏ nhất, *fmin* sẽ ước lượng hàm để tìm giá trị này. Quá trình tìm kiếm sẽ tốn thời gian nếu như hàm có một lượng phép tính lớn, hoặc là hàm có nhiều hơn một giá trị cực tiểu trong dải tìm kiếm. Trong một số trường hợp, quá trình này không tìm ra được đáp số. Khi mà *fmin* không tìm được giá trị nhỏ nhất thì nó dừng lại và đưa ra lời giải thích.

16.3 Tìm giá trị không

Nếu như bạn đã quan tâm đến việc tìm kiếm khi hàm tiến ra vô cùng, thì đó, khi lại cần tìm ra khi nào hàm qua 0 và khi nào qua các giá trị không đổi

MATLAB cung cấp cho ta công cụ để giải quyết vấn đề này. Hàm *fzero* tìm giá trị 0 của mảng một chiều. Để làm sáng tỏ, chúng ta cùng xem lại ví dụ về hàm *humps* một lần nữa:

```

>> xzero = fzero('humps',1.2), % look for zero near 1.2
xzero =
1.2995
>> yzero = humps(xzero) % evaluate at zero
yzero =
3.5527e-15

```

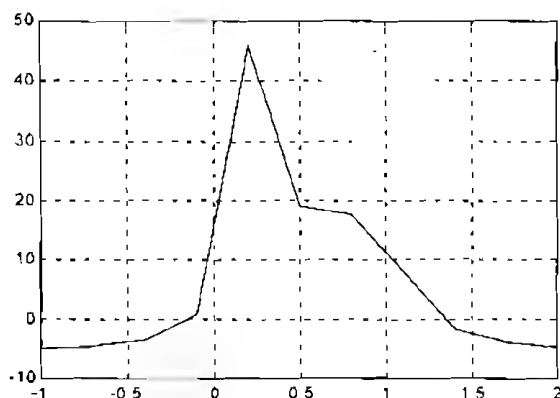
Như vậy, giá trị 0 gần với 1.3. Như thấy ở trên, quá trình tìm kiếm giá trị 0 có thể không có kết quả. Nếu không tìm thấy, nó dừng lại và đưa ra giải thích.

Hàm *fzero* bắt buộc phải được cung cấp tên cho nó mỗi khi nó được gọi đến. *fzero* cho biết tại đâu hàm bằng 0 hoặc nó còn có thể tìm ra giá trị để kn. nào hàm bằng hằng số. Ví dụ tìm x để $f(x)=c$, thì ta phải định nghĩa lại hàm $g(x)$ như sau: $g(x)=f(x)-c$, và hàm *fzero* tìm giá trị của x để $g(x)=0$, tương đương $f(x)=c$.

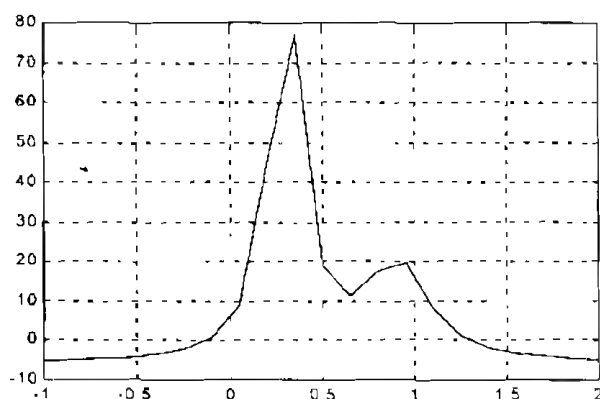
16.4 Phép lấy tích phân

MATLAB cung cấp cho ta ba hàm để tính các phép toán liên quan đến tích phân: *trapz*, *quad* và *quad8*. Hàm *trapz* cho ta giá trị xấp xỉ tích phân ở phía dưới hàm bằng cách lấy tổng các miền hình thang của các điểm dữ liệu như trong hình 16.3.

Như thấy trong hình 16.3, các miền hình thang độc lập có giá trị ước lượng dưới mức thực tế. Nếu ta chia nhỏ ra như phép nội suy tuyến tính thì sự xấp xỉ của hàm sẽ cao hơn. Ví dụ nếu ta gấp đôi số lượng các hình thang đã có, thì đồ xấp xỉ tăng lên như hình vẽ 16.4.



Hình 16.3



Hình 16.4

Tính toán các vùng này bằng hàm $y = \text{humps}(x)$ với $-1 < x < 2$, sử dụng *trapz* cho mỗi hình trên ta có:

```
>> x = -1:3/2;    % rough approximation
>> y = humps(x);
>> area = trapz(x,y) % call trapz just like the plot command
area =
    21.8453
>> x = -1:1/15:2; % better approximation
>> y = humps(x);
>> area = trapz(x,y)
area =
    25.8523
```

Thông thường thì kết quả của chúng là khác nhau, dựa trên số lượng các miền được chia trong hình vẽ. Tuy nhiên, không có gì đảm bảo rằng quá trình xấp xỉ nào là tốt hơn, ngoại trừ sự đúng đắn của phép toán, nên nhiên khi bạn thay đổi một cách độc lập các vùng hình thang, vì như làm cho nó nhỏ đi thì chắc chắn là kết quả sẽ chính xác hơn nhiều.

Hàm *quad* và *quad8* đều là các hàm có cách tính như nhau. Sự định giá của cả hai hàm là rất cần thiết để đạt kết quả chính xác. Hơn nữa độ xấp xỉ của chúng là cao hơn so với hình thang đơn, với *quad8* có kết quả chính xác hơn *quad*. Các hàm này được gọi giống như gọi *fzero*.

```
>> area = quad('humps',-1,2) % find area between -1 and 2
area =
    26.3450
>> area = quad8('humps',-1,2)
area =
    26.3450
```

Để biết thêm chi tiết về hàm này, bạn hãy xem trên hệ trợ giúp của MATLAB.

16.5 Phép lấy vi phân

So sánh với phép lấy tích phân, ta thấy phép lấy vi phân khó hơn nhiều. Phép lấy tích phân cho cả một vùng, hoặc đặc tính vĩ mô của hàm trong khi phép lấy vi phân chỉ lấy tại một điểm

nào đấy, hay còn gọi là đặc tính vi mô của hàm. Kết quả là phép tính vi phân sẽ không ổn định khi đặc tính của hình thay đổi trong khi phép tính tích phân thì ít chịu ảnh hưởng hơn.

Bởi vì phép tính tích phân là khó nên người ta cố tránh những phép tính nào mà không thể thực hiện được, đặc biệt khi dữ liệu lấy tích phân là kết quả của thực nghiệm. Ví dụ, chúng ta hãy xem xét ví dụ làm trơn hình trong chương 15:

```
>> x = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1];  
>> y = [-.447 1.978 3.28 6.16 7.08 7.34 ...  
        7.66 9.56 9.48 9.30 11.2]; % data  
>> n = 2; % order of fit  
>> p = polyfit(x,y,n) % find polynomial coefficients  
p =  
   -9.8108   20.1293   -0.0317  
>> xi = linspace(0,1,100);  
>> z = polyval(p,xi); % evaluate polynomial  
>> plot(x,y,'o',x,y,xi,z,':')  
>> xlabel('x'),ylabel('y=f(x)')  
>> title('Second Order Curve Fitting')
```

Vi phân trong trường hợp này được sử dụng bằng cách sử dụng hàm đạo hàm *polyder*:

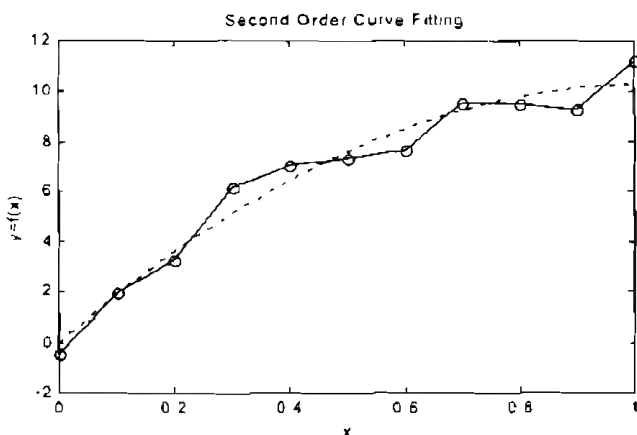
```
>> pd = polyder(p)  
pd =  
   -19.6217   20.1293
```

Vi phân của đa thức $y = -9.8108x^2 + 20.1293x - 0.0317$ là

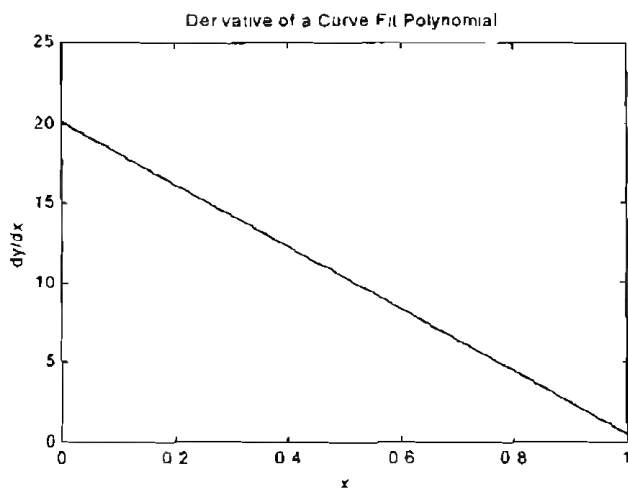
$$dx/dy = -19.6217x + 20.1293$$

Bởi vì đạo hàm của một đa thức cũng được vẽ và tính giá trị giống như là đối với đa thức:

```
>> z = polyval(pd,xi); % evaluate derivative  
>> plot(xi,z)  
>> xlabel('x'),ylabel('dy/dx')  
>> title('Derivative of a Curve Fit Polynomia')
```



Hình 16.5



Hình 16.6

Trong trường hợp này xấp xỉ đa thức là một hàm bậc hai và đạo hàm của nó trở thành hàm bậc nhất (hình 16.6).

MATLAB cung cấp một hàm để tính toán đạo hàm một cách sơ bộ dựa vào dữ liệu mô tả một số hàm, hàm này có tên là *diff*, nó tính toán độ chênh lệch giữa các phần tử trong mảng. Bởi vì đạo hàm được định nghĩa như sau:

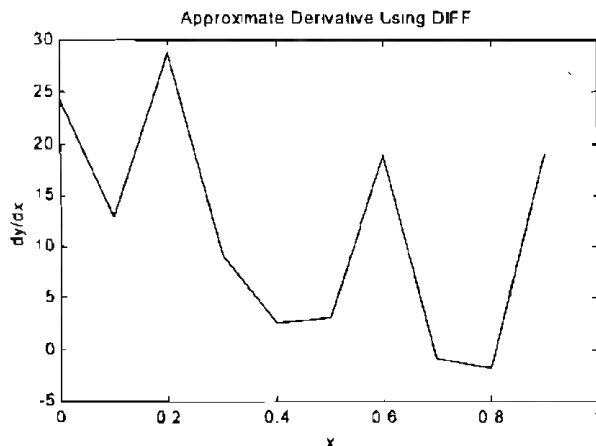
$$\frac{dy}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{(x+h) - x}$$

nên đạo hàm của hàm $f(x)$ có thể được tính một cách sơ bộ dựa vào công thức.

$$\frac{dy}{dx} = \frac{f(x+h) - f(x)}{(x+h) - x} \quad \text{khi } h > 0$$

Gọi là số ra của y chia cho số ra của x, do hàm **diff** tính toán sự khác nhau giữa các phần tử trong mảng nên đạo hàm có thể được tính một cách xấp xỉ dựa vào hàm **diff**.

```
>> dy = diff(y)./diff(x);
>> % compute differences and use array division
>> xd = x(1:length(x)-1);
>> % create new x axis array since dy is shorter than y
>> plot(xd,dy)
>> title('Approximate Derivative Using DIFF')
>> ylabel('dy/dx'),xlabel('x')
```



Hình 16.7

Do hàm **diff** tính ra sự khác nhau giữa các phần tử nên kết quả của ví dụ trên là một mảng có số phần tử ít hơn mảng ban đầu một phần tử. Vì vậy để vẽ được đồ thị của đạo hàm thì phải bỏ đi một phần tử của mảng x. So sánh hai đồ thị cuối cùng (hình 16.6 và 16.7) thì thấy hiển nhiên rằng đạo hàm tính bằng phương pháp gần đúng khác xa so với thực tế.

16.6 Phương trình vi phân

Có thể bạn đã khá quen với thực tế là rất nhiều hệ thống vật lý đều được mô tả bằng phương trình vi phân. Do vậy phần sau đây đối với bạn có thể khá hấp dẫn.

Một phương trình vi phân thường mô tả tốc độ thay đổi của một biến số trong hệ thống theo sự thay đổi của một biến khác trong hệ thống hoặc theo kích thích bên ngoài. Phương trình

vi phân thông thường có thể được giải nhờ các phương pháp giải tích hoặc sử dụng công cụ toán kí hiệu của MATLAB.

Trong những trường hợp mà phương trình vi phân không thể giải được bằng phương pháp giải tích thì việc sử dụng phương pháp số học trở lên khá hiệu quả. Để minh hoạ hãy xét phương trình Van Der Pol, phương trình biểu diễn một bộ dao động:

$$\frac{d^2x}{dt^2} - \mu(1-x^2)\frac{dx}{dt} + x = 0$$

Tất cả các phương pháp toán học để giải phương trình dạng này đều sử dụng một phương trình vi phân cao cấp hơn, tương đương với một tập phương trình vi phân bậc nhất. Đối với phương trình vi phân trên thì cách giải này được thực hiện bằng cách định nghĩa hai biến trung gian:

$$\text{đặt } y_1 = x, \text{ và } y_2 = \frac{dx}{dt}$$

$$\text{suy ra: } \frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = \mu(1-y_1^2) - y_2$$

Đối với các hệ phương trình như thế này MATLAB cung cấp một tập các hàm ODE để giải xấp xỉ hoá chúng một cách số học. Trong cuốn sách này chúng tôi không đề cập hết những nội dung và ứng dụng của từng hàm trong bộ ODE. Để tìm hiểu thêm về các hàm ODE ứng dụng trong rất nhiều bài toán thí dụ, hãy gõ `>> odedemo` tại dấu nhắc của MATLAB. Trước hết chúng ta hãy xét ví dụ sau đây (chính là ví dụ *ode45*). Chúng ta phải viết một hàm `M_file` trả về các đạo hàm nếu biết trước các giá trị tức thời của y_1 và y_2 . Trong MATLAB các đạo hàm được cho bởi các vector cột, trong trường hợp này gọi là *yprime*. Tương tự y_1 và y_2 được viết dưới dạng vector cột y . Kết quả của một hàm `M_file` như sau:

```
function yprime=vdpo(t,y);
% VDPOL(t,y) returns the state derivatives of
% the Van der Pol equation:
%
%  $x'' - \mu(1-x^2)x' + x = 0$ 
%
% let  $y(1)=x$  and  $y(2)=x'$ 
```

```
%
% then y(1)'=y(2)
% y(2)'=mu*(1-y(1)^2)*y(2)-y(1)
mu=2; % choose 0< mu < 10
yprime=[y(2)
mu*(1-y(1)^2)*y(2)-y(1)]; % output must be a column
```

Giả sử thời gian kéo dài từ 0 đến 30 giây, ví dụ `tspan=[0 30]`. Sau đó sử dụng lệnh ***vdpol*** thì lời giải cho bài toán như sau:

```
>> tspan = [0 30];
>> yo = [1;0];
>> ode45('vdpol',tspan,yo);
```

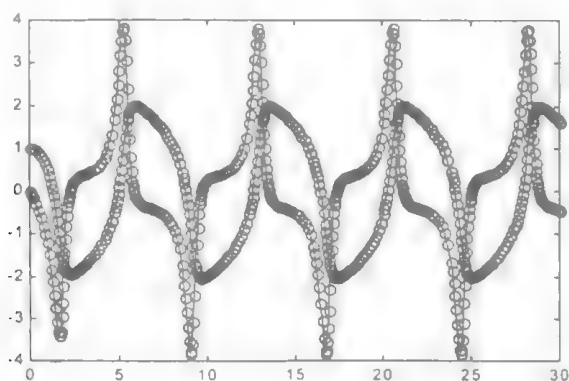
Khi sử dụng hàm mà không có đối số ra, các hàm ODE sẽ tự động chọn những thời điểm thích hợp để tính đạo hàm. Để có thể truy nhập được dữ liệu, ta chỉ cần cung cấp cho hàm những thông số ra.

```
>> [t,y] = ode45('vdpol',tspan,yo);
```

Ở đây `t` là một vector cột chứa những thời điểm để tính đạo hàm, còn `y` là một ma trận chứa hai cột và các hàng `length(t)`, hàng đầu tiên của ma trận `y` chứa biến số `y(1)`, hàng thứ hai là biến số `y(2)`.

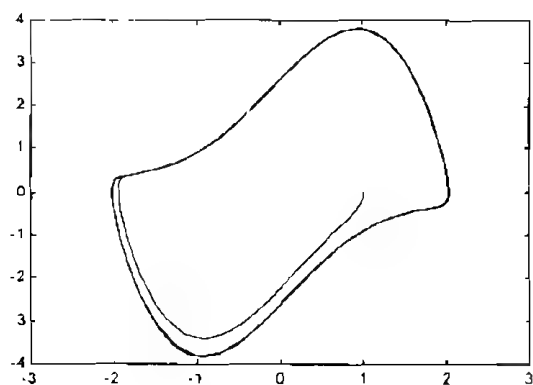
Dựa vào những đặc điểm này chúng ta có thể vẽ được đồ thị pha, là đồ thị giữa `y(2)` và `y(1)` (hình 16.8):

```
>> plot(y(:,1),y(:,2))
```



Hình 16.8

Các hàm ODE của MATLAB đều có trợ giúp trực tuyến, mỗi hàm đều có các đối số cũng như cách sử dụng riêng, nếu bạn muốn nghiên cứu thêm thì hãy tham khảo thêm phần trợ giúp trực tuyến của chúng.



Hình 16.9

ĐỒ HOẠ TRONG HỆ TOẠ ĐỘ PHẪNG

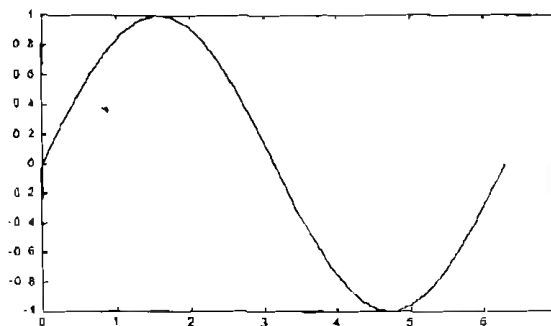
Trong phần này giới thiệu một số đặc tính về đồ hoạ của MATLAB.

17.1 Sử dụng lệnh Plot

Như bạn đã thấy ở ví dụ mục 16, phần lớn các câu lệnh để vẽ đồ thị trong mặt phẳng đều là lệnh **plot**. Lệnh **plot** này sẽ vẽ đồ thị của một mảng dữ liệu trong một hệ trục thích hợp, và nối các điểm bằng đường thẳng. Dưới đây là một ví dụ mà bạn đã thấy trước đó (hình 17.1):

```
>> x = linspace(0,2*pi,30);
>> y = sin(x);
>> plot(x,y)
```

Ví dụ này tạo 30 điểm dữ liệu trong đoạn $0 \leq x \leq 2\pi$ theo chiều ngang đồ thị, và tạo một vector y khác là hàm sine của dữ liệu chứa trong x. Lệnh **plot** mở ra một cửa sổ đồ hoạ gọi là cửa sổ **figure**, trong cửa sổ này nó sẽ tạo độ chia phù hợp với dữ liệu vẽ đồ thị qua các điểm, và đồ thị được tạo thành bởi việc nối các điểm này bằng đường nét liền. Các thang chia số và dấu được tự động cập nhật vào, nếu như cửa sổ **figure** đã tồn tại, **plot** xoá cửa sổ hiện thời và thay vào đó là cửa sổ mới.

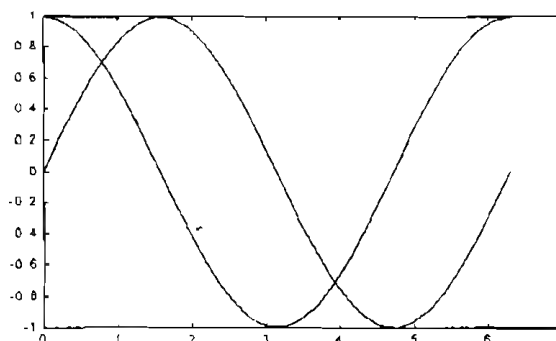


Hình 17.1

Bây giờ cùng vẽ hàm sine và cosine trên cùng một đồ thị (hình 7.2):

```
>> z = cos(x);
```

```
>> plot(x,y,x,z)
```



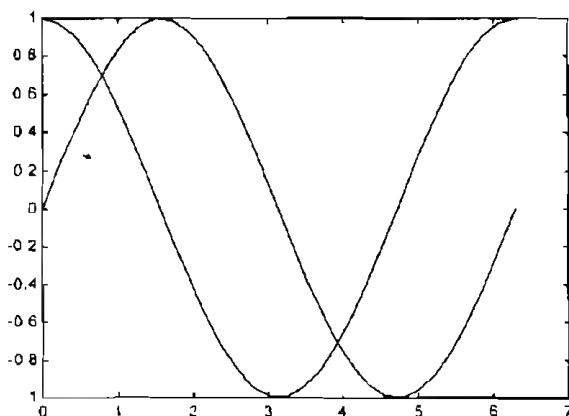
Hình 17.2

Ví dụ này cho thấy bạn có thể vẽ nhiều hơn một đồ thị trên cùng một hình vẽ, bạn chỉ việc đưa thêm vào *plot* một cặp đối số, *plot* tự động vẽ đồ thị thứ hai bằng màu khác trên màn hình. Nhiều đường cong có thể cùng vẽ một lúc nếu như bạn cung cấp đủ các cặp đối số cho lệnh *plot*.

Nếu như một trong các đối số là ma trận và đối số còn lại là vector, thì lệnh *plot* sẽ vẽ tương ứng mỗi cột của ma trận với vector đó:

```
>> W = [y;z] % xây dựng một ma trận sine và cosine
```

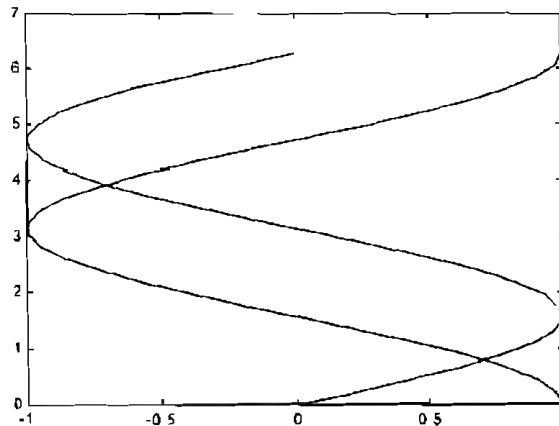
```
>> plot(x,W) % vẽ các cột của W với x
```



Hình 17.3

Nếu như bạn thay đổi trật tự các đối số thì đồ thị sẽ xoay một góc bằng 90 độ.

```
>> plot(W,x)
```



Hình 17.4

Nếu lệnh **plot** được gọi mà chỉ có một đối số, ví như **plot(Y)** thì hàm **plot** sẽ đưa ra một kết quả khác, phụ thuộc vào dữ liệu chứa trong Y. Nếu giá trị của Y là một số phức, **plot(Y)** tương đương với **plot(real(Y))** và **plot(imag(Y))**, trong tất cả các trường hợp khác thì phần ảo của Y thường được bỏ qua. Mặt khác nếu Y là phần thực thì **plot(Y)** tương ứng với

plot(1:length(Y), Y).

17.2 Kiểu đường, dấu và màu

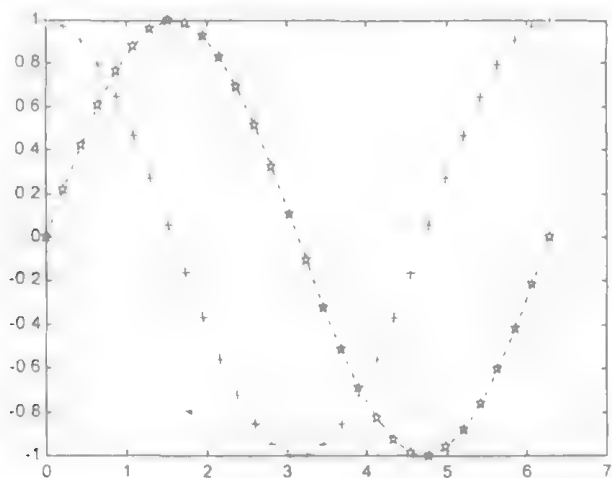
Trong ví dụ trước, MATLAB chọn kiểu nét vẽ **solid** và màu **blue** và **green** cho đồ thị. Ngoài ra bạn có thể khai báo kiểu màu, nét vẽ của riêng của bạn bằng việc đưa vào **plot** một đối số thứ 3 sau mỗi cặp dữ liệu của mảng. Các đối số tùy chọn này là một xâu kí tự, có thể chứa một hoặc nhiều hơn theo bảng dưới đây.

Nếu bạn không khai báo mẫu thì MATLAB sẽ chọn màu mặc định là **blue**. Kiểu đường mặc định là kiểu **solid** trừ khi bạn khai báo kiểu đường khác. Còn về dấu, nếu không có dấu nào được chọn thì sẽ không có kiểu của dấu nào được vẽ.

Nếu một màu, dấu, và kiểu đường tất cả đều chứa trong một xâu, thì kiểu màu chung cho cả dấu và kiểu nét vẽ. Để khai báo màu khác cho dấu, bạn phải vẽ cùng một dữ liệu với các kiểu khai báo chuỗi khác nhau. Dưới đây là một ví dụ sử dụng các kiểu đường, màu, và dấu vẽ khác nhau:

```
>> plot(x,y,'b:p',x,z,'c-',x,z,'m+')
```

Biểu tượng	Màu	Biểu tượng	Dấu	Biểu tượng	Kiểu nét vẽ
b	xanh da trời	.	điểm	-	nét liền
g	xanh lá cây	o	tròn	:	đường chấm
r	đỏ	x	dấu x	-.	đường gạch-chấm
c	xanh xám	+	dấu +	--	đường gạch-gạch
m	đỏ tím	*	sao		
y	vàng	s	vuông		
k	đen	d	diamond		
w	trắng	v	triangle(down)		
		^	triangle (up)		
		<	triangle(left)		
		>	triangle(right)		
		p	pentagram		
		h	hexagram		



Hình 17.5a

17.3 Kiểu đồ thị

Lệnh **colordel** cho phép bạn lựa chọn kiểu hiển thị. Giá trị mặc định của **colordel** là **white**. Kiểu này sử dụng trục tọa độ, màu nền, nền hình vẽ màu xám sáng, và tên tiêu đề của

trục màu đen. Nếu bạn thích nền màu đen, bạn có thể dùng lệnh ***colordel black***. Kiểu này sẽ cho ta nền trục tọa độ đen, nền hình vẽ màu tối xám, và tiêu đề trục màu trắng.

17.4 Đồ thị lưới, hộp chứa trục, nhãn, và lời chú giải

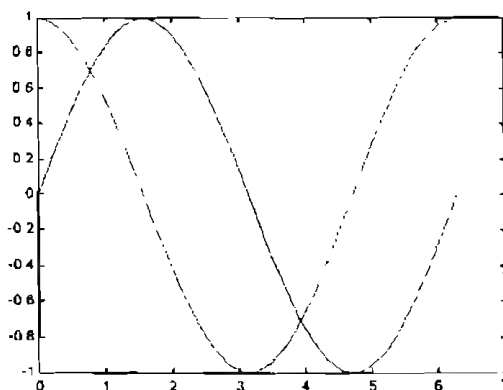
Lệnh ***grid on*** sẽ thêm đường lưới vào đồ thị hiện tại (hình 17.7). Lệnh ***grid off*** sẽ bỏ các nét này, lệnh ***grid*** mà không có tham số đi kèm theo thì sẽ xen kẽ giữa chế độ ***on*** và ***off***. MATLAB khởi tạo với ***grid off***. Thông thường trục tọa độ có nét gấn kiểu ***solid*** nên gọi là hộp chứa trục. Hộp này có thể tắt đi với ***box off*** và ***box on*** sẽ khôi phục lại. Trục đứng và trục ngang có thể có nhãn với lệnh ***xlabel*** và ***ylabel***. Lệnh ***title*** sẽ thêm vào đồ thị tiêu đề ở đỉnh (hình 17.6). Dùng hàm ***sine*** và ***cosine*** để minh họa:

```
>> x = linspace(0,2*pi,30);
```

```
>> y = sin(x);
```

```
>> z = cos(x);
```

```
>> plot(x,y,x,z)
```



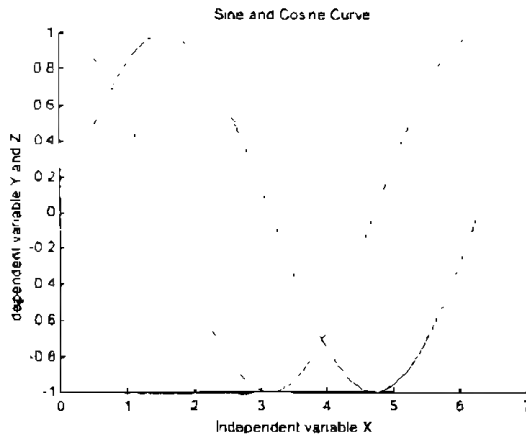
Hình 17.5b

```
>> box off
```

```
>> xlabel('Independent variable X')
```

```
>> ylabel('dependent variable Y and Z')
```

```
>> title('Sine and Cosine Curve')
```



Hình 17.6

Bạn có thể thêm nhãn hoặc bất cứ chuỗi kí tự nào vào bất cứ vị trí nào bằng cách sử dụng lệnh **text**. Cú pháp của lệnh này là: **text(x, y, 'string')** trong đó x, y là toạ độ tâm bên trái của chuỗi văn bản. Để thêm nhãn vào hình **sine** ở vị trí (2.5, 0.7) như sau:

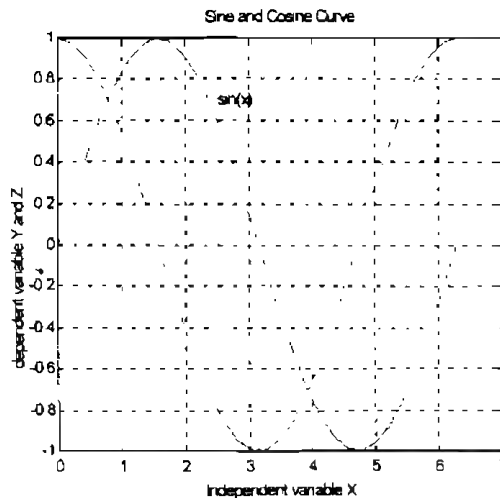
```
>> grid on, box on
```

```
>> text(2.5,0.7,'sin(x)')
```

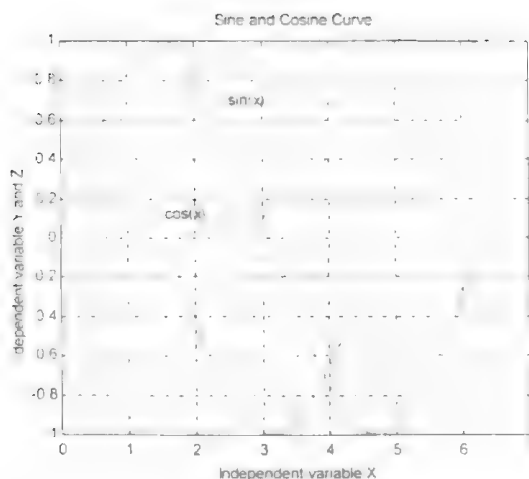
Nếu bạn muốn thêm nhãn mà không muốn bỏ hình vẽ khỏi hệ trục đang xét, bạn có thể thêm chuỗi văn bản bằng cách di chuột đến vị trí mong muốn. Lệnh **gtext** sẽ thực hiện việc này.

Ví dụ (Hình 17.8):

```
>> gtext('cos(x)')
```



Hình 17.7



Hình 17.8

17.5 Kiến tạo hệ trục tọa độ

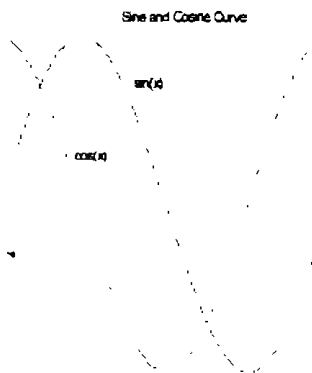
MATLAB cung cấp cho bạn công cụ để có thể kiểm soát hoàn toàn hình dáng và thang chia của cả hai trục đứng và ngang với lệnh **axis**. Do lệnh này có nhiều yếu tố, nên chỉ một số dạng hay dùng nhất được đề cập ở đây. Để biết một cách đầy đủ về lệnh **axis**, bạn hãy xem hệ trợ giúp **help** của MATLAB hoặc các tham khảo khác. Các đặc tính cơ bản của lệnh **axis** được cho trong bảng dưới đây:

Lệnh	Mô tả
<code>axis([xmin xmax ymin ymax])</code>	Thiết lập các giá trị min, max của hệ trục dùng các giá trị được đưa ra trong vector hàng
<code>V=axis</code>	V là một vector cột có chứa thang chia cho đồ thị hiện tại: [xmin xmax ymin ymax]
<code>axis auto</code>	Trả lại giá trị mặc định thang chia
<code>axis ('auto')</code>	<code>xmin = min(x)</code> , <code>xmax = max(x)</code> , ...v.v...
<code>axis manual</code>	Giới hạn thang chia như thang chia hiện tại
<code>axis xy</code>	Sử dụng (mặc định) hệ tọa độ decac trong đó góc tọa độ ở góc góc thấp nhất bên trái, trục ngang tăng từ trái qua phải, trục đứng tăng từ dưới lên trên.

axis ij	Sử dụng hệ tọa độ ma trận, trong đó gốc tọa độ ở đỉnh góc trái, trục đứng tăng từ đỉnh xuống, trục ngang tăng từ trái qua phải.
axis square	Thiết lập đồ thị hiện tại là hình vuông, so với mặc định hình chữ nhật
axis equal	Thiết lập thang chia giống nhau cho cả hai hệ trục
axis tightequal	Tương tự như axis equal nhưng hộp đồ thị vừa đủ đối với dữ liệu
axis normal	Tắt đi chế độ axis equal, equal, tight và vis3d
axis off	Tắt bỏ chế độ nền trục, nhãn, lưới, và hộp, dấu. Thoát khỏi chế độ lệnh title và bất cứ lệnh label nào, và thay bởi lệnh text và gtext
axis on	Ngược lại với axis off nếu chúng có thể.

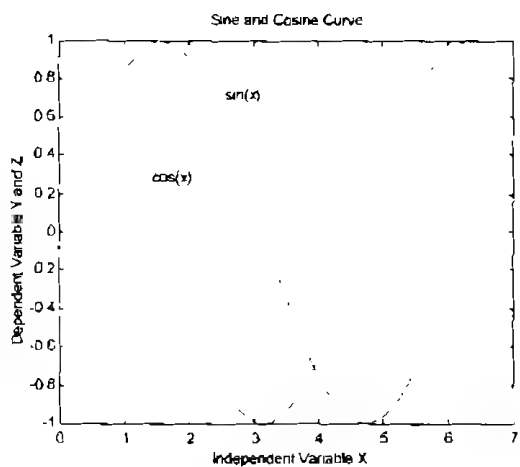
Thử kiểm nghiệm một số lệnh **axis** cho đồ thị của bạn, sử dụng các ví dụ trước đó sẽ cho ta kết quả như sau:

```
>> axis off % bỏ trục tọa độ
```



Hình 17.9

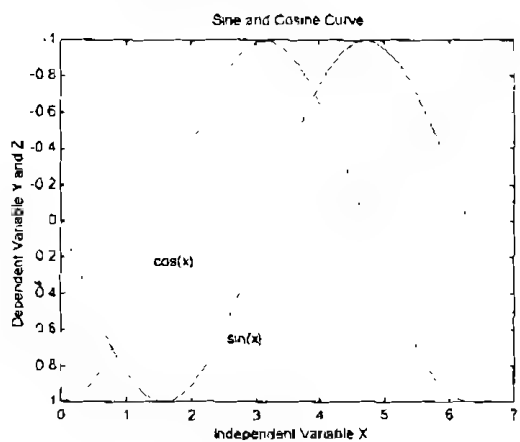
```
>> axis on, grid off % turn the axis on, the grid off
```



Hình 17.10

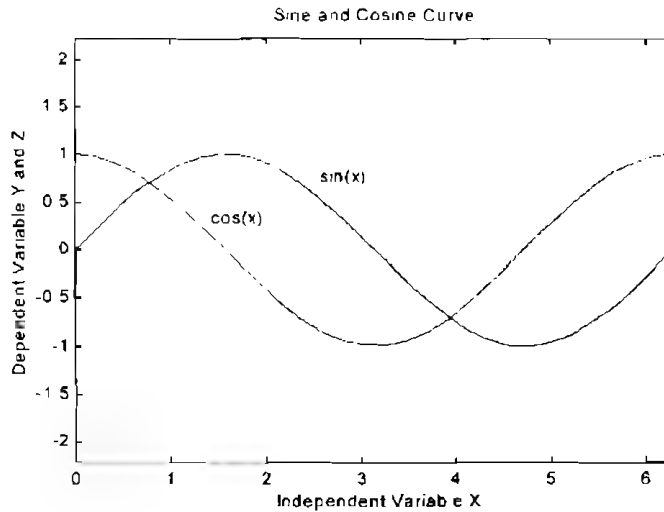
>>axis ij

% turn the plot upside-down



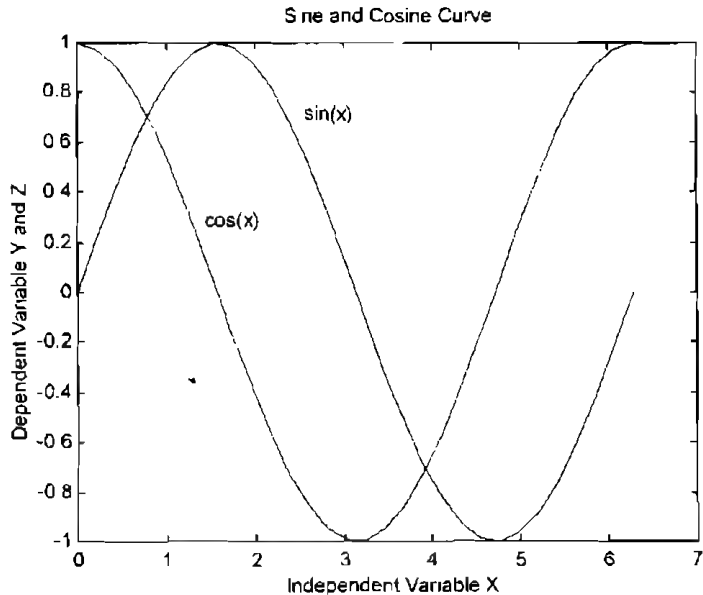
Hình 17.11

>> axis square equal % give axis two command at once



Hình 17.12

>> axis xy normal % return to the defaults



Hình 17.13

17.6 In hình

Để in các hình mà bạn vừa vẽ hoặc các hình trong chương trình của MATLAB mà bạn cần, bạn có thể dùng lệnh in từ bảng chọn hoặc đánh lệnh in vào từ cửa sổ lệnh:

- In bằng lệnh từ bảng chọn: Trước tiên ta phải chọn cửa sổ hình là cửa sổ hoạt động bằng cách nhấn chuột lên nó, sau đó bạn chọn mục bảng chọn **Print** từ bảng chọn **file**. Dùng các thông số tạo lên trong mục bảng chọn **Print Setup** hoặc **Page Setup**, đồ thị hiện tại của bạn sẽ được gửi ra máy in.

- In bằng lệnh từ cửa sổ lệnh. Trước tiên bạn cũng phải chọn cửa sổ hình làm cửa sổ hoạt động bằng cách nhấn chuột lên nó hoặc dùng lệnh *figure(n)*, sau đó bạn dùng lệnh in.

```
>> print % prints the current plot to your printer
```

Lệnh *orient* sẽ thay đổi kiểu in: Kiểu mặc định là kiểu *portrait*, in theo chiều đứng, ở giữa trang. Kiểu in *landscape* là kiểu in ngang và kín toàn bộ trang. Kiểu in *tall* là kiểu in đứng nhưng kín toàn bộ trang. Để thay đổi kiểu in khác với kiểu mặc định, bạn dùng lệnh *orient* với các thông số của nó như sau:

```
>> orient % What is the current orientation
```

```
ans=
```

```
portrait
```

```
>> orient landscape % print sideways on the page
```

```
>> orient tall % stretch to fill the vertical page
```

Nếu bạn muốn tìm hiểu kỹ hơn về chúng thì hãy xem trợ giúp trực tuyến về chúng.

17.7 Thao tác với đồ thị

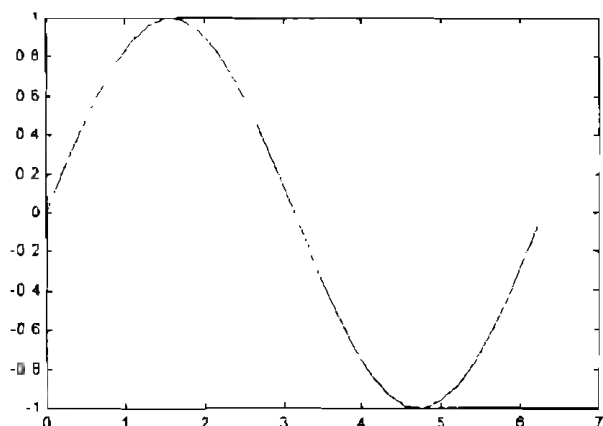
Bạn có thể thêm nét vẽ vào đồ thị đã có sẵn bằng cách dùng lệnh *hold*. Khi bạn thiết lập *hold on*, MATLAB không bỏ đi hệ trục đã tồn tại trong khi lệnh *plot* mới đang thực hiện, thay vào đó, nó thêm đường cong mới vào hệ trục hiện tại. Tuy nhiên nếu như dữ liệu không phù hợp với hệ trục tọa độ cũ, thì trục được chia lại. Thiết lập *hold off* sẽ bỏ đi cửa sổ *figure* hiện tại và thay vào bằng một đồ thị mới. Lệnh *hold* mà không có đối số sẽ bật tắt chức năng của chế độ thiết lập *hold* trước đó. Trở lại với ví dụ trước:

```
>> x = linspace(0,2*pi,30);
```

```
>> y = sin(x);
```

```
>> z = cos(x);
```

```
>> plot(x,y)
```



Hình 17.14

Bây giờ giữ nguyên đồ thị và thêm vào đường cosine

```
>> hold on
```

```
>> ishold % hàm logic này trả về giá trị 1 (true) nếu hold ở trạng thái ON
```

```
ans =
```

```
1
```

```
>> plot(x,z,'m')
```

```
>> hold off
```

```
>> ishold % hold bây giờ không còn ở trạng thái ON nữa.
```

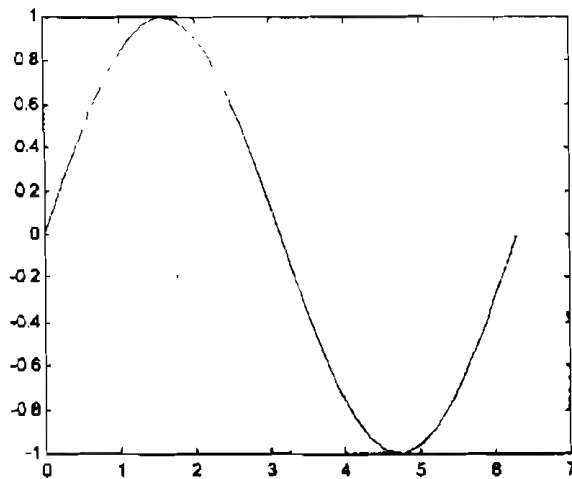
```
ans =
```

```
0
```

Chú ý rằng để kiểm tra trạng thái của *hold* ta có thể dùng hàm *ishold*.

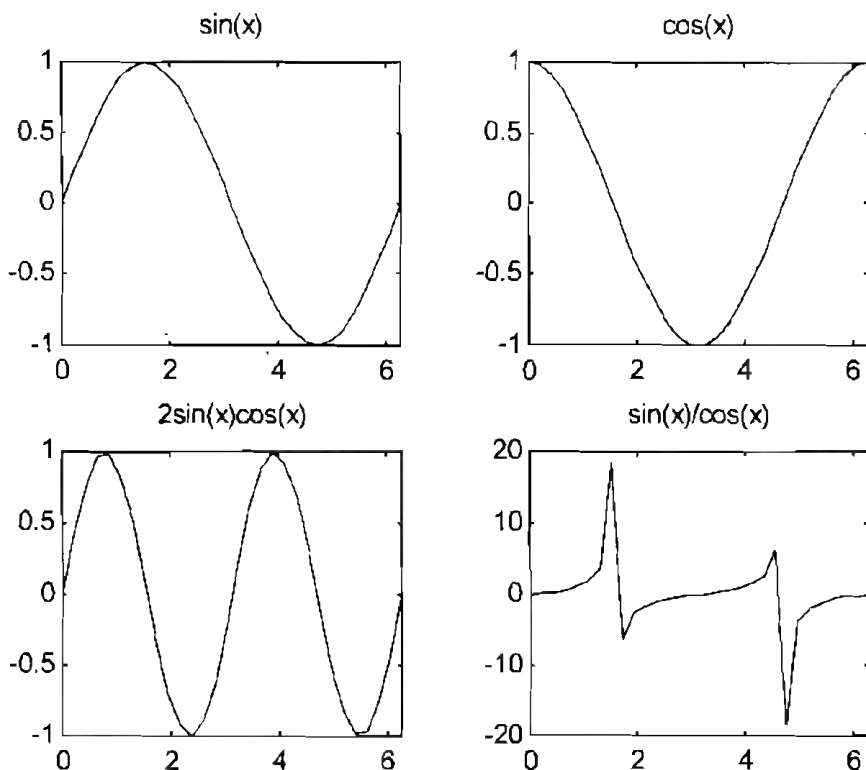
Nếu bạn muốn hai hay nhiều đồ thị ở các cửa sổ *figure* khác nhau, hãy dùng lệnh *figure* trong cửa sổ lệnh hoặc chọn *new figure* từ bảng chọn file, *figure* không có tham số sẽ tạo một *figure* mới. Bạn có thể chọn kiểu *figure* bằng cách dùng chuột hoặc dùng lệnh *figure(n)* trong đó *n* là số cửa sổ hoạt động.

Mặt khác một cửa sổ *figure* có thể chứa nhiều hơn một hệ trục. Lệnh *subplot(m,n,p)* chia cửa sổ hiện tại thành một ma trận *m* x *n* khoảng để vẽ đồ thị, và chọn *p* là cửa sổ hoạt động. Các đồ thị thành phần được đánh số từ trái qua phải, từ trên xuống dưới, sau đó đến hàng thứ hai v.v... Ví dụ:



Hình 17.15

```
>> x = linspace(0,2*pi,30);
>> y = sin(x);
>> z = cos(x);
>> a = 2*sin(x).*cos(x);
>> b = sin(x)./(cos(x)+eps);
>> subplot(2,2,1) % pick the upper left of
    % 2 by 2 grid of subplots
>> plot(x,y),axis([0 2*pi -1 1]),title('sin(x)')
>> subplot(2,2,2) % pick the upper right of the 4 subplots
>> plot(x,z),axis([0 2*pi -1 1]),title('cos(x)')
>> plot(x,z),axis([0 2*pi -1 1]),title('cos(x)')
>> subplot(2,2,3)% pick the lower left of the 4 subplots
>> plot(x,a),axis([0 2*pi -1 1]),title('2sin(x)cos(x)')
>> subplot(2,2,4)%pick the lower right of the 4 subplots
>> plot(x,b),axis([0 2*pi -20 20]),title('sin(x)/cos(x)')
```

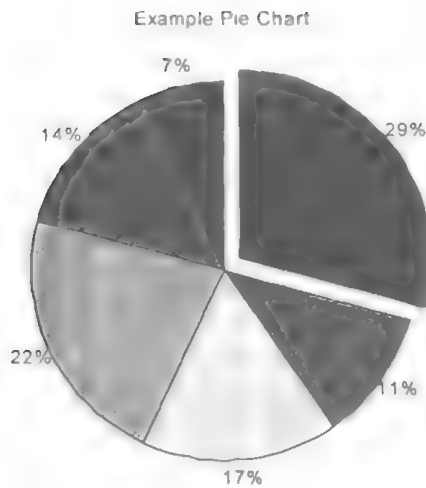


Hình 17.6

17.8 Một số đặc điểm khác của đồ thị trong hệ tọa độ phẳng

- **loglog** tương tự như **plot** ngoại trừ thang chia là **logarithm** cho cả hai trục.
- **semilogx** tương tự như **plot** ngoại trừ thang chia của trục x là **logarithm** còn thang chia trục y là tuyến tính.
- **semilogy** tương tự như **plot** ngoại trừ thang chia của trục y là **logarithm**, còn thang chia trục x là tuyến tính.
- **area(x, y)** tương tự như **plot(x,y)** ngoại trừ khoảng cách giữa 0 và y được điền đầy, giá trị cơ bản y có thể được khai báo, nhưng mặc định thì không.
- Sơ đồ hình múi tiêu chuẩn được tạo thành từ lệnh **pie(a, b)**, trong đó **a** là một vector giá trị và **b** là một vector logic tùy chọn (hình 17.7). Ví dụ:

```
>> a = [.5 1 1.6 1.2 .8 2.1];
>> pie(a,a==max(a));
>> title('Example Pie Chart')
```

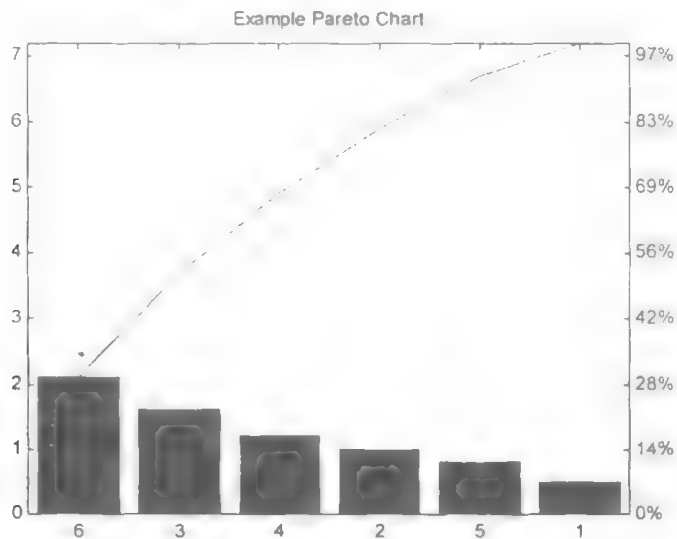


Hình 17.7

▪ Một cách khác để quan sát dữ liệu đó là biểu đồ **Pareto**, trong đó các giá trị trong các vector được vẽ thành một khối chữ nhật (hình 17.8). Ví dụ dùng vector *a* đã nói ở trên:

```
>> pareto(a);
```

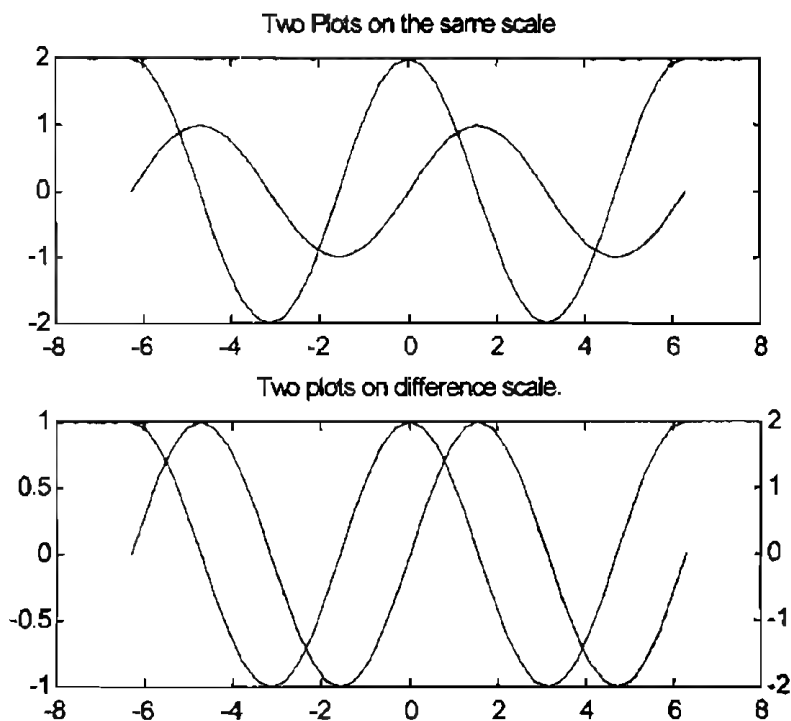
```
>> title('Example Pareto Chart')
```



Hình 17.18

▪ Đôi khi bạn muốn vẽ hai hàm khác nhau trên cùng một hệ trục mà lại sử dụng thang chia khác nhau, **plotyy** có thể làm điều đó cho bạn:

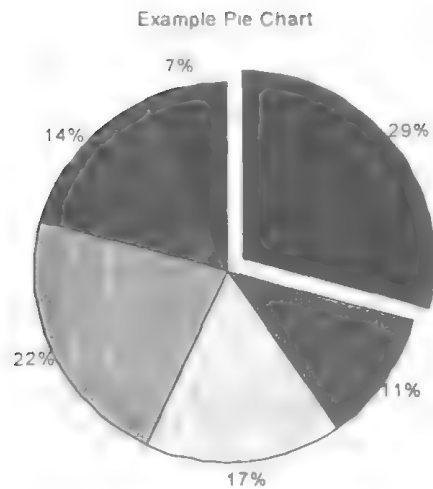
```
>> x = -2*pi/10:2*pi;
>> y = sin(x); z = 2*cos(x);
>> subplot(2,1,1),plot(x,y,x,z),
>> title('Two Plots on the same scale');
>> subplot(2,1,2),plotyy(x,y,x,z)
>> title('Two plots on difference scale.');
```



Hình 17.19

Đồ thị **bar** và **stair** có thể sinh ra bởi việc dùng lệnh **bar**, **bar3**, **barh** và **stairs**. Dưới đây là ví dụ:

```
>> x = -2.9:0.2:2.9;
>> y = exp(-x.*x);
```

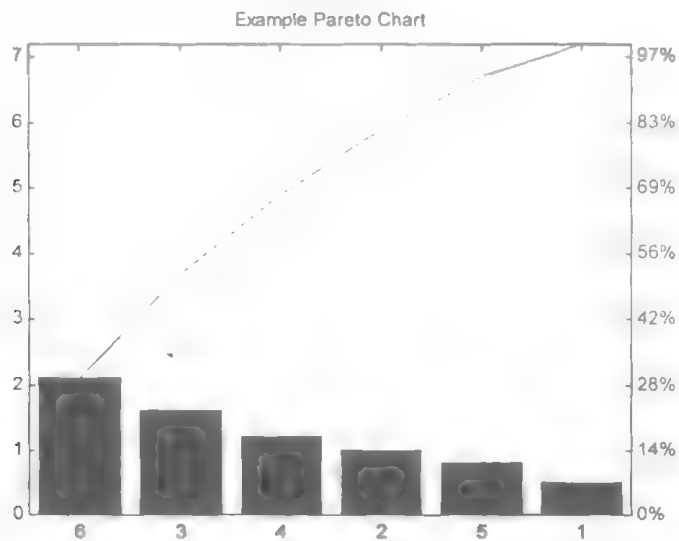


Hình 17.7

- Một cách khác để quan sát dữ liệu đó là biểu đồ *Pareto*, trong đó các giá trị trong các vector được vẽ thành một khối chữ nhật (hình 17.8). Ví dụ dùng vector *a* đã nói ở trên:

```
>> pareto(a);
```

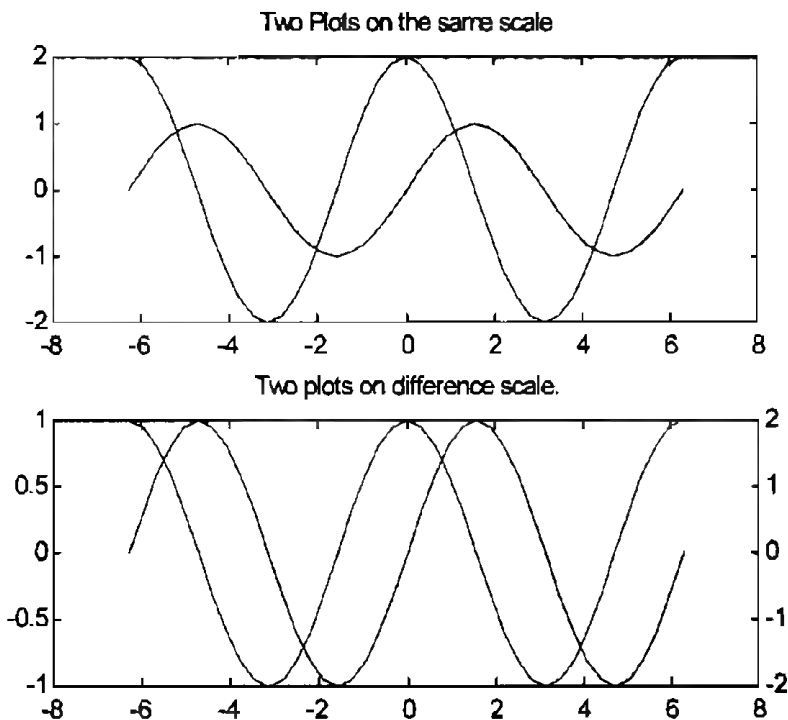
```
>> title('Example Pareto Chart')
```



Hình 17.18

▪ Đôi khi bạn muốn vẽ hai hàm khác nhau trên cùng một hệ trục mà lại sử dụng thang chia khác nhau, *plotyy* có thể làm điều đó cho bạn:

```
>> x = -2*pi/pi/10:2*pi;  
>> y = sin(x); z = 2*cos(x);  
>> subplot(2,1,1),plot(x,y,x,z),  
>> title('Two Plots on the same scale');  
>> subplot(2,1,2),plotyy(x,y,x,z)  
>> title('Two plots on difference scale.');
```



Hình 17.19

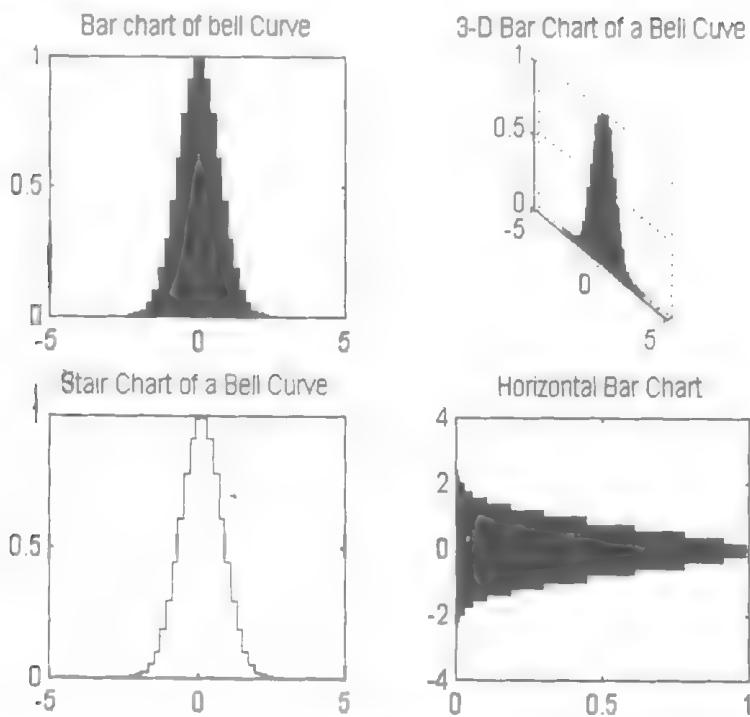
Đồ thị *bar* và *stair* có thể sinh ra bởi việc dùng lệnh *bar*, *bar3*, *barh* và *stairs*. Dưới đây là ví dụ:

```
>> x = -2.9:0.2:2.9;  
>> y = exp(-x.*x);
```

```

>> subplot(2,2,1)
>> bar(x,y)
>> title('Bar chart of bell Curve')
>> subplot(2,2,2)
>> bar3(x,y)
>> title('3-D Bar Chart of a Bell Cuve')
>> subplot(2,2,3)
>> stairs(x,y)
>> title('Stair Chart of a Bell Curve')
>> subplot(2,2,4)
>> barh(x,y)
>> title('Horizontal Bar Chart')

```

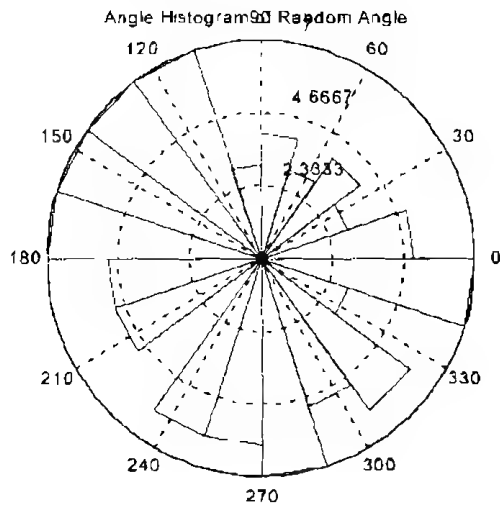


Hình 17.20

- `rose(V)` vẽ một biểu đồ trong tọa độ cực cho các góc trong vector `v`, tương tự ta cũng có các

lệnh `rose(v,n)` và `rose(v,x)` trong đó `x` là một vector. Dưới đây là một ví dụ:

```
>> v = randn(100,1)*pi;
>> rose(v)
>> title('Angle Histogram of Random Angle')
```



Hình 17.21

ĐỒ HOẠ TRONG KHÔNG GIAN BA CHIỀU

MATLAB cung cấp một số hàm để hiển thị dữ liệu 3 chiều như các hàm vẽ đường thẳng trong không gian 3 chiều, các hàm vẽ bề mặt và khung dây và màu có thể được sử dụng thay thế cho chiều thứ tư.

18.1 Đồ thị đường thẳng

Lệnh *plot* từ trong không gian hai chiều có thể mở rộng cho không gian 3 chiều bằng lệnh *plot3*. Khuôn dạng của *plot3* như sau:

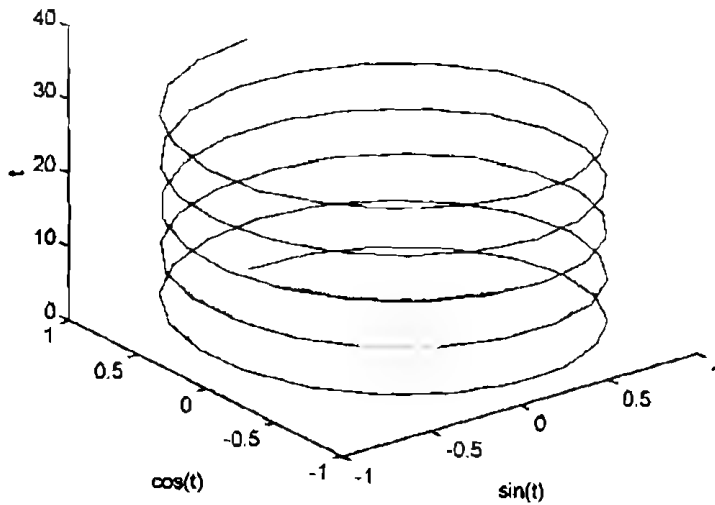
plot3(x1, y1, z1, S1, x2, y2, z2, S2,), trong đó x_n , y_n và z_n là các vector hoặc ma trận, và S_n là chuỗi ký tự tùy chọn dùng cho việc khai báo màu, tạo biểu tượng hoặc kiểu đường. Sau đây là một số ví dụ:

```
>> t = linspace(0, 10*pi);
>> plot3(sin(t), cos(t), t)
>> title('Helix'), xlabel('sin(t)')
>> ylabel('cos(t)'), zlabel('t') % Hình 18.1
```

Chú ý rằng: hàm *zlabel* tương ứng với hàm hai chiều *xlabel* và *ylabel*. Tương tự như vậy, lệnh *axis* cũng có khuôn dạng: *axis([xmin xmax ymin ymax zmin zmax])* thiết lập giới hạn cho cả 3 trục. Ví dụ:

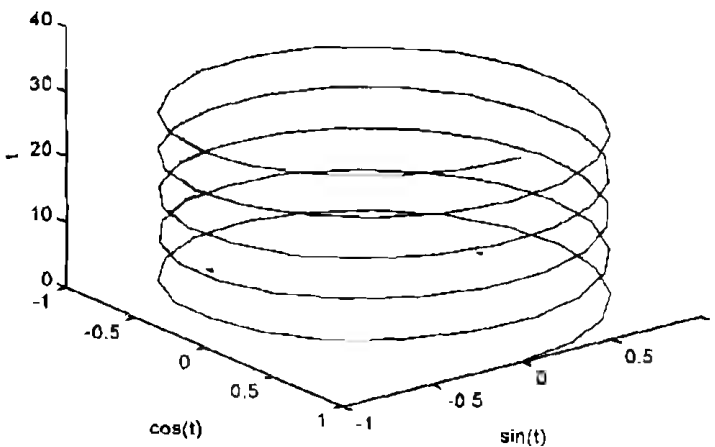
```
>> axis('ij') % thay đổi hướng trục từ sau ra trước (hình 18.2)
```

Helix



Hình 18.1 Đường Helix trong không gian ba chiều

Helix



Hình 18.2

Hàm `text` cũng có khuôn mẫu như sau: `text(x, y, z, 'string')` sẽ đặt vị trí xâu 'string' vào tọa độ x, y, z .

18.2 Đồ thị bề mặt và lưới

MATLAB định nghĩa bề mặt lưới bằng các điểm theo hướng trục z ở trên đường kẻ ô hình vuông trên mặt phẳng x - y . Nó tạo lên mẫu một đồ thị bằng cách ghép các điểm gần kề với các đường thẳng. Kết quả là nó trông như một mạng lưới đánh cá với các mắt lưới là các điểm dữ liệu. Đồ thị lưới này thường được sử dụng để quan sát những ma trận lớn hoặc vẽ những hàm có hai biến.

Bước đầu tiên là đưa ra đồ thị lưới của hàm hai biến $z = f(x, y)$, tương ứng với ma trận X và Y chứa các hàng và các cột lặp đi lặp lại. MATLAB cung cấp hàm **meshgrid** cho mục đích này. $[X, Y] = \text{meshgrid}(x, y)$, tạo một ma trận X , mà các hàng của nó là bản sao của vector x , và ma trận Y có các cột của nó là bản sao của vector y . Cặp ma trận này sau đó được sử dụng để ước lượng hàm hai biến sử dụng đặc tính toán học về mảng của MATLAB.

Sau đây là một ví dụ về cách dùng hàm **meshgrid**.

```
>> x = -7.5:5:7.5;
```

```
>> y = x;
```

```
>> [X,Y] = meshgrid(x,y);
```

X, Y là một cặp của ma trận tương ứng một lưới chữ nhật trong mặt phẳng x - y . Mọi hàm $z=f(x,y)$ có thể sử dụng tính chất này.

```
>> R = sqrt(X.^2+Y.^2)+eps;
```

```
>> % find the distance from the origin (0,0)
```

```
>> Z = sin(R)./R; % calculate sin(r)/r
```

Ma trận R chứa bán kính của mỗi điểm trong $[X,Y]$, nó là khoảng cách từ mỗi điểm đến tâm ma trận. Cộng thêm **eps** để không để xảy ra phép chia cho 0. Ma trận Z chứa sine của bán kính chia cho bán kính mỗi điểm trong sơ đồ. Câu lệnh sau vẽ đồ thị lưới:

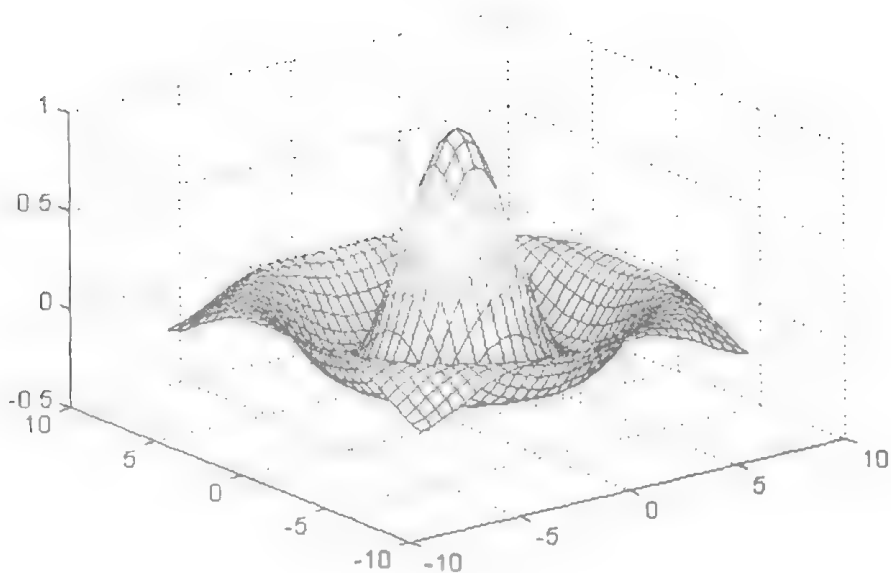
```
>> mesh(X,Y,Z)
```

Đồ thị là đơn sắc (hình 18.3). Tuy nhiên bạn có thể thay đổi màu sắc với sự trợ giúp của MATLAB rất dễ dàng nếu bạn đọc đến phần **colormaps**.

Trong ví dụ này, hàm **mesh** sắp xếp giá trị của các phần tử của ma trận vào các điểm (X_i, Y_i, Z_i) trong không gian ba chiều. **mesh** cũng có thể vẽ một ma trận đơn tương tự như với một đối số: **mesh(Z)**, sử dụng các điểm (i,j,Z_{ij}) . Như vậy Z được vẽ ngược lại với các chỉ số của nó, trong trường hợp này **mesh(Z)** chỉ đơn giản là chia lại độ khắc các trục x, y theo các chỉ số của ma trận Z . Bạn hãy thử tạo ví dụ cho trường hợp này?

Đồ thị bề mặt của cùng một ma trận Z trông như đồ thị lưới trước đó, ngoại trừ khoảng cách giữa hai đường là khác nhau (gọi là **patches**). Đồ thị loại này dùng hàm **surf**, nó có tất cả các đối số như hàm **mesh**. Hãy xem ví dụ dưới đây (hình 18.4):

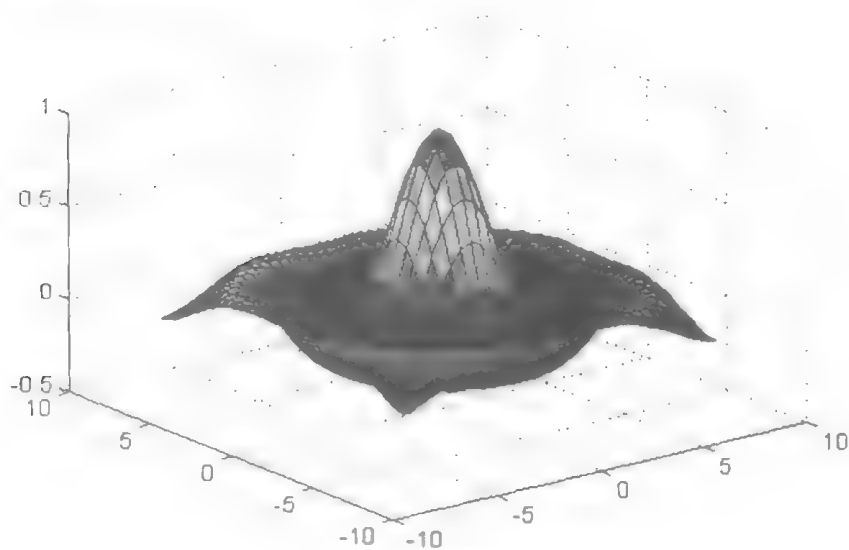
```
>> surf(X,Y,Z)
```



Hình 18.3

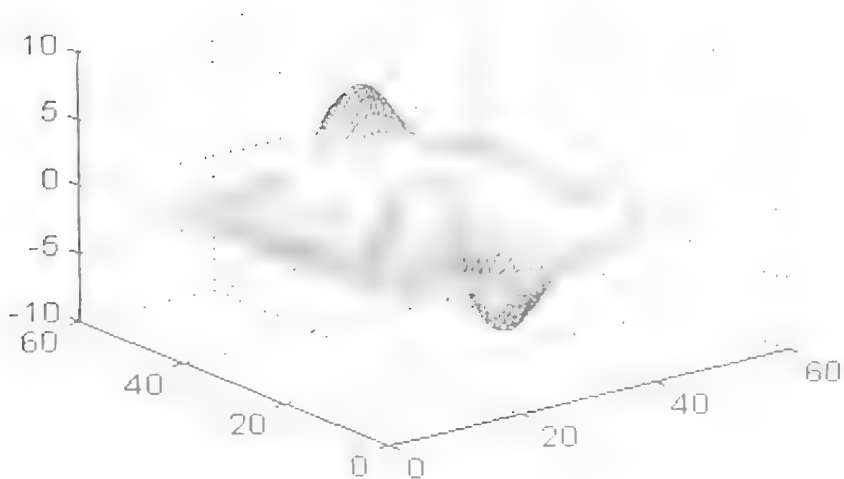
Để làm rõ thêm một vài chủ đề, chúng ta cùng quay lại hàm **peaks** đã đưa ra ở phần trước. Đồ thị lưới trong không gian 3 chiều của hàm này được đưa ra như sau (hình 18.5):

```
>> mesh(peaks)
>> title('Mesh Plot of Peaks function')
```



Hình 18.4

mesh plot of the peaks function



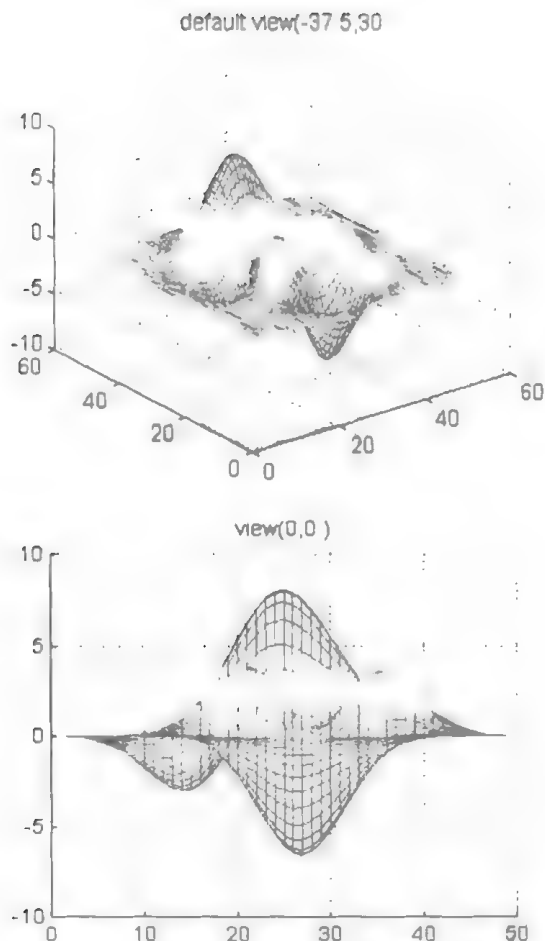
Hình 18.5

Đồ thị đường viền cho ta thấy được độ năng hoặc độ cao của hình. Trong MATLAB đồ thị đường viền trong không gian hai chiều tương tự như trong không gian ba chiều nhưng hàm gọi của nó là `contour3`. Đồ thị sử dụng các lệnh sẽ được minh họa trong bảng khác màu.

18.3 Thao tác với đồ thị

MATLAB cho phép bạn khai báo góc để từ đó quan sát được đồ thị trong không gian ba chiều. Hàm `view(azimuth, elevation)` thiết lập góc xem bằng việc khai báo **azimuth** và **elevation**. “Elevation” mô tả vị trí người quan sát, xem như là góc đo bằng độ trên hệ trục x-y. “Azimuth” mô tả góc trong hệ trục nơi người quan sát đứng.

Azimuth được đo bằng độ từ phần âm trục y. Phía âm trục y có thể quay theo chiều kim đồng hồ một góc -37.5 độ từ phía ban. **Elevation** là góc mà tại đó mắt bạn thấy được mặt phẳng x-y. Sử dụng hàm `view` cho phép bạn có thể quan sát hình vẽ từ các góc độ khác nhau. Ví dụ nếu **elevation** thiết lập là âm, thì `view` sẽ nhìn hình từ phía dưới lên. Nếu **azimuth** thiết lập dương, thì hình sẽ quay ngược chiều kim đồng hồ từ điểm nhìn mặc định. Thêm chỉ bạn có thể nhìn trực tiếp từ trên bằng cách thiết lập `view(0,90)`. Thực ra thì đây là điểm nhìn mặc định 2 chiều, trong đó x tăng từ trái qua phải, và y tăng từ trên xuống dưới, khuôn dạng `view(2)` hoàn toàn giống như mặc định của `view(0, 90)`, và `view(3)` thiết lập mặc định trong không gian 3 chiều. Ví dụ:



Hình 18.6

Lệnh **view** có một dạng khác mà rất tiện ích khi sử dụng là **view([X,Y,Z])** cho phép bạn quan sát trên một vector chứa hệ trục tọa độ decac trong không gian 3 chiều. Khoảng cách từ vị trí bạn quan sát đến góc tọa độ không bị ảnh hưởng. Ví dụ, **view([0 10 0])**, **view([0 -1 0])** và **view(0, 0)** cho các kết quả như nhau. Các thông số **azimuth** và **elevation** mà bạn đang quan sát có thể lấy lại được bằng cách dùng **[az, e] = view**. Ví dụ:

```
>> view([-7 -9 7])
```

```
>> [az,e] = view
```

```
az =
```

```
-37.8750
```

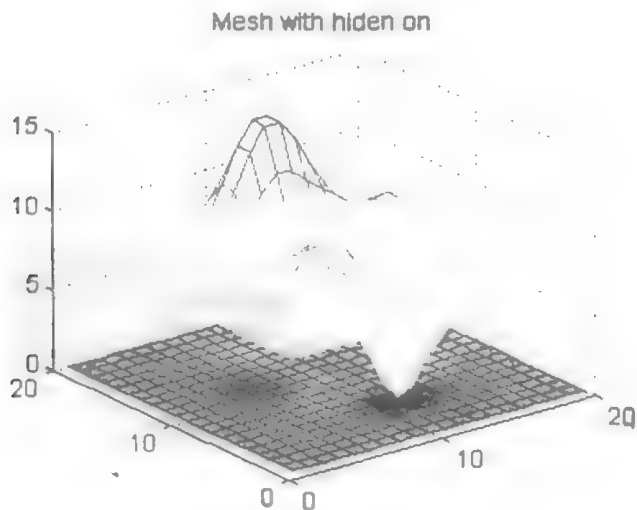
el =

31.5475

Một công cụ hữu dụng khác là quan sát đồ thị không gian 3 chiều bởi hàm **rotate3d**. Các thông số **Azimuth** và **elevation** có thể được tác động bởi chuột, **rotate3d on** cho phép chuột can thiệp, **rotate3d off** không cho phép.

Lệnh **hidden** dấu các nét khuất. Khi bạn vẽ đồ thị, thì một số phần của nó bị che khuất bởi các phần khác, khi đó nếu dùng lệnh này thì các nét khuất sẽ bị dấu đi, bạn chỉ có thể nhìn phần nào ở trong tầm nhìn của bạn. Nếu bạn chuyển đến **hidden off**, bạn có thể thấy phần khuất đó qua mạng lưới. Dưới đây là ví dụ:

```
>> mesh(peaks(20)+7)
>> hold on
>> pcolor(peaks(20))
>> hold off
>> title('Mesh with hidden on')
```

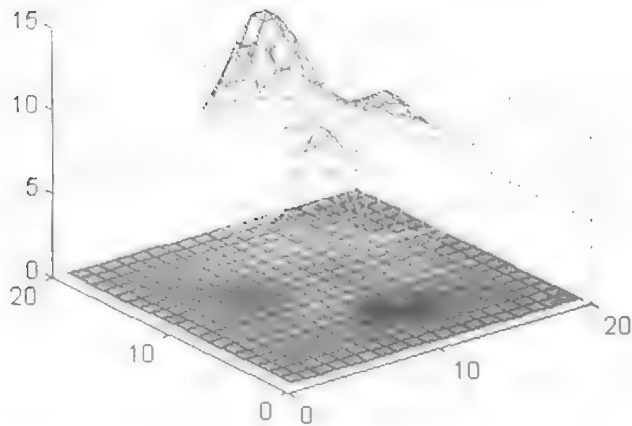


Hình 18.7

Bây giờ hãy bỏ chế độ dấu các nét khuất đi ta sẽ thấy sự khác nhau:

```
>> hidden off
>> title('Mesh with Hidden Off')
```

Mesh with hidden off

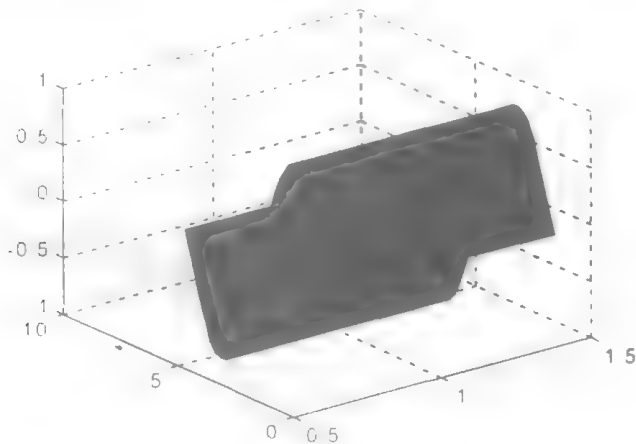


Hình 18.9

18.4 Các đặc điểm khác của đồ thị trong không gian 3 chiều

- Hàm *ribbon(x, y)* tương tự như *plot(x, y)* ngoại trừ cột của y được vẽ như là một dải riêng biệt trong không gian ba chiều. Trên hình 18.10 là đồ thị hình sin.

Ribbon Plot of a sine Curve



Hình 18.10

- Hàm *clabel* tăng thêm độ cao cho đồ thị đường viền. Có ba mẫu *clabel(cs)*, *clabel(cs, V)* và *clabel(cs, 'manual')* *clabel(cs)*, trong đó *cs* là cấu trúc đường viền được trả về từ lệnh *contour*, *cs=contour(z)*, lấy nhận tất cả các đồ thị đường viền với độ cao của nó. Vì

trị của nhân được lấy ngẫu nhiên **clabel** (c. 'manual') định vị nhân đường viền ở vị trí kích chuột tương tự như lệnh **ginput** đã nói ở trên. Nhân phím **Return** kết thúc việc tạo nhân này

- Hàm **contourf** sẽ vẽ một đồ thị đường viền kín, không gian giữa đường viền được lấp đầy bằng màu.

- Hai mẫu trạng thái của lệnh **mesh** dung với đồ thị lưới là **meshc** vẽ đồ thị lưới và thêm đường viền bên dưới, **meshz** vẽ đồ thị lưới và đồ thị co dạng như màn che.

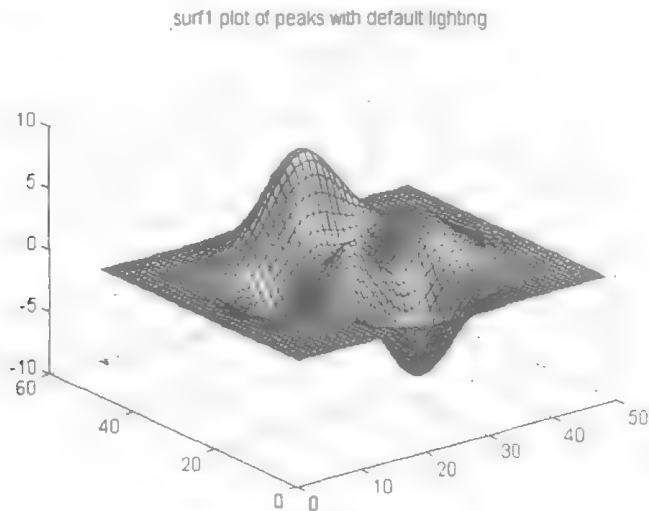
- Hàm **waterfall** được xem như **mesh** ngoại trừ một điều là hàm **mesh** chỉ xuất hiện ở hướng x.

- Có hai mẫu trạng thái của lệnh **surf**, đó là **surfc** vẽ một đồ thị **surf** và thêm đường bao bên dưới, **surflvex** vẽ một đồ thị **surf** nhưng thêm vào sự chiếu sáng bề mặt từ nguồn sáng. Cấu trúc tổng quát là **surfl(X,Y, Z, S, K)** trong đó X, Y, và Z tương tự như **surf**, S là một vector tùy chọn trong hệ tọa độ decac ($S=[S_x \ S_y \ S_z]$) hoặc trong tọa độ cầu ($S=[az,el]$) chỉ ra hướng của nguồn sáng. Nếu không khai báo, giá trị mặc định của S là 45 độ theo chiều kim đồng hồ từ vị trí người quan sát, S là một vector tùy chọn chỉ ra phần đóng góp tùy thuộc vào nguồn sáng bao quanh, sự phản chiếu ánh sáng và hệ số phản chiếu ($K=[ka,kd,ks,spread]$)

```
>> colormap(gray)
```

```
>> surfl(peaks)
```

```
>> title('surf1 plot of peaks with default lighting')
```



Hình 18.11

- **fill3**, phiên bản 3 chiều của **fill**, vẽ một đa giác đều trong không gian ba chiều. Khung dạng tổng quát của nó là **fill3(x, y, z, c)**, trong đó chiều đứng của đa giác được chỉ bởi ba thanh

phân x, y, z . Nếu c là một kí tự, đa giác sẽ được lấp đầy màu như ở bảng màu. c cũng có thể là một vector hàng có 3 thành phần ($[r\ g\ b]$) trong đó r, g và b là các giá trị giữa 0 và 1 thay cho các màu đỏ, xanh lá cây và xanh da trời. Nếu c là một vector hoặc ma trận, nó được sử dụng như một chỉ số chỉ ra sơ đồ màu. Nhiều đa giác có thể được tạo ra bằng cách cho thêm nhiều đối số như *fill3* ($x1, y1, z1, c1, x2, y2, z2, c2, \dots$). Ví dụ sau sẽ vẽ ngẫu nhiên 4 tam giác với màu:

```
>> color(cool)
```

```
>> fill3(rand(3,4),rand(3,4),rand(3,4),rand(3,4))
```

- *bar3* và *bar3h* là phiên bản 3 chiều của *bar* và *barh*, *bie3* là phiên bản của *pie*.

18.5 Bảng màu

Màu và biểu đồ màu được đề cập đến trong một số ví dụ ở phần trước. Trong phần này chúng ta sẽ nói rõ về chúng. MATLAB định nghĩa một biểu đồ màu như là một ma trận có 3 cột. Mỗi hàng của ma trận định nghĩa một màu riêng biệt sử dụng các số trong dải 0 và 1. Những số này chỉ ra các giá trị RGB, độ nhạy của các màu thành phần đỏ, xanh lá cây, và xanh da trời trong một màu do các thành phần đỏ tạo ra. Một số mẫu cơ bản được cho trong bảng dưới đây:

Đỏ	Xanh lá cây	Xanh da trời	Màu
0	0	0	đen
1	1	1	trắng
1	0	0	đỏ
0	1	0	xanh lá cây
0	0	1	xanh da trời
1	1	0	vàng
1	0	1	tím đỏ
0	1	1	lam xám
-5	-5	-5	xám trung bình
-5	0	0	đỏ tối
1	-62	-40	đỏ đồng
-49	1	-83	ngọc xanh biển

Dưới đây là một số hàm của MATLAB để tạo ra bảng màu ở trên:

Function	Mô tả bảng màu
hsv	Giá trị màu bão hoà (HSV)
hot	đen-đỏ-vàng-trắng
gray	xám cân bằng tuyến tính
bone	xám có pha nhẹ với màu xanh
copper	sắc thái của màu đồng
pink	màu hồng nhạt nhẹ
white	trắng hoàn toàn
flag	xen kẽ đỏ, trắng, xanh da trời, và đen
jet	sự thay đổi màu bão hoà
prism	có màu sắc lăng kính
cool	màu xanh tím
lines	màu của nét vẽ
summe	Bóng của xanh lá cây và vàng
autumn	Bóng của đỏ và vàng
winter	Bóng của xanh lá cây và xanh da trời
spring	Bóng của magenta và yellow

18.6 Sử dụng bảng màu

Câu lệnh *colormap(M)* cài đặt ma trận *M* như là bảng màu được sử dụng bởi hình hiện tại.

Ví dụ *colormap(cool)* cài đặt một version 64 đầu vào của bảng màu *cool*.

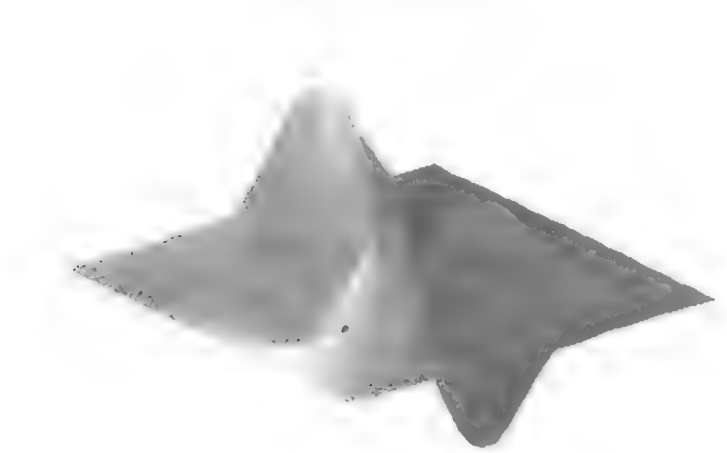
Hàm *plot* và *plot3* không dùng bảng màu ở trên, chúng sử dụng các màu liệt kê trong bảng kiểu dương, điểm đánh dấu, màu của *plot*. Phần lớn các hàm vẽ khác như *mesh*, *surf*, *contour*, *fill*, *pcolor* và các biến của nó, sử dụng bảng màu hiện tại.

Sau đây là một ví dụ dùng tham số màu cho hàm *surf* để hiển thị góc quan sát

```
>> [X,Y,Z]=peaks(30);
>> surf(X,Y,Z,atan2(X,Y))
```

```
>> colormap(hsv),shading flat
>> axis([-3 3 -3 3 -6.5 8.1]),axis off
>> title('using a color Argument to surf')
```

using a color Argument to surf



Hình 18.12

18.7 Sử dụng màu để thêm thông tin

Màu có thể được dùng để thêm thông tin vào đồ thị 3 chiều nếu nó được sử dụng để tạo thành chiều thứ tư. Các hàm như *mesh* và *surf* biến đổi màu dọc theo trục z, trừ khi một đối số màu được đưa ra như *surf(X,Y,Z)* hoàn toàn tương đương với *surf(X,Y,Z,t)* trong đó thành phần thứ t được dùng như một chỉ số trong biểu đồ màu. Điều này khiến cho đồ thị đầy màu nhưng lại không thông tin khi mà trục z đã tồn tại.

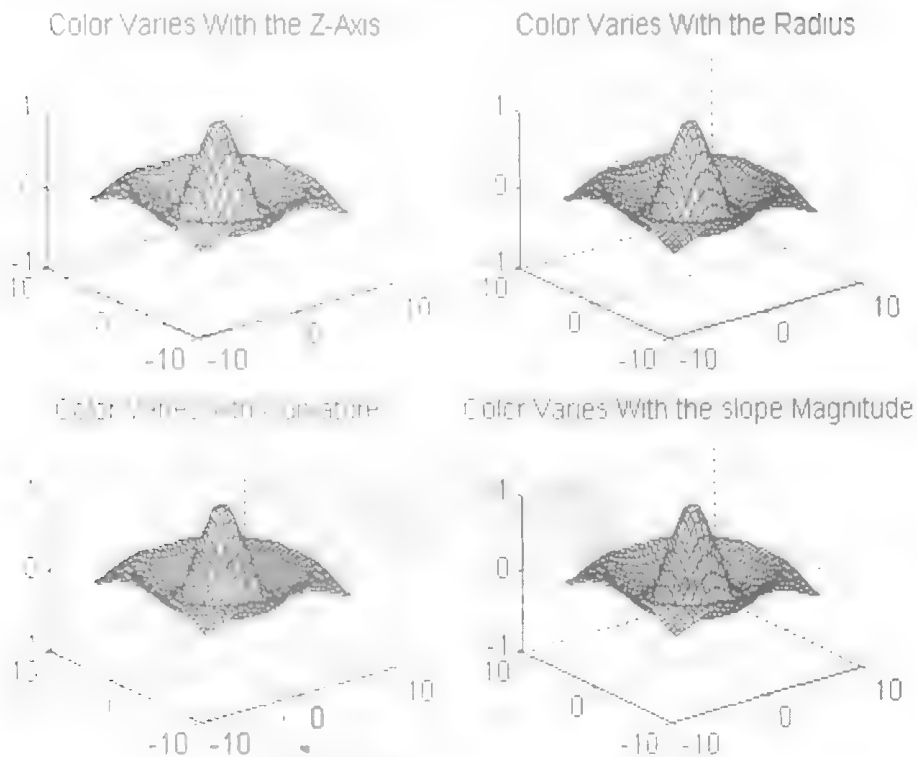
Dưới đây là một số cách sử dụng đối số màu để thêm thông tin hoặc nhấn mạnh thông tin đã tồn tại trong đồ thị

```
>> x=-7.5: .5:7.5; y=x % create a data set
>> [X,Y]=meshgrid(x,y); %create plaid data
>> R=sqrt(X.^2+Y.^2) +eps % create radial data
>> Z=sin(R)./R; % create a sombrero
>> subplot(2,2,1),surf(X,Y,Z),
>> title('Color Varies with the Z_axis')
>> subplot(2,2,2),surf(X,Y,Z,R),
```

```

>> title('Color Varies With the Radius')
>> subplot(2,2,3),surf(X,Y,Z,dZ),
>> title('Color Varies with Curvature')
>> [dZdx,dZdy]=gradient(Z); %compute the slope
>> dZ=sqrt(dZdx.^2+dZdy.^2) %compute the slope's manitude
>> subplot(2,2,4),surf(X,Y,Z,dZ)
>> title('Color Varies With the slope Magnitude')

```



Hình 18.13

18.8 Hiện thị bằng màu

Bạn có thể hiện thị bằng màu theo một số cách sau. Một trong những cách đó là xem tất cả các phân tử trong một ma trận bằng màu một cách trực tiếp

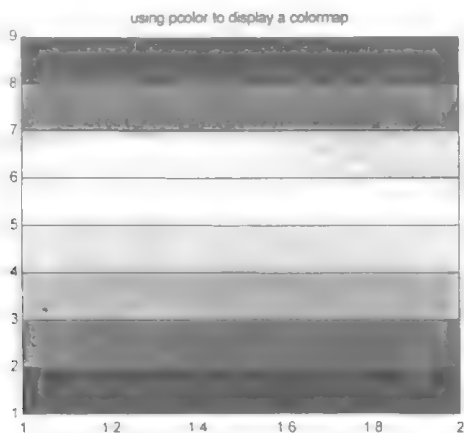
```
>> hot(8)
```

ans =

0.3333	0	0
0.6667	0	0
1.0000	0	0
1.0000	0.3333	0
1.0000	0.6667	0
1.0000	1.0000	0
1.0000	1.0000	0.5000
1.0000	1.0000	1.0000

Thêm vào đó, hàm *pcolor* có thể được sử dụng để biểu diễn một bảng màu. Hãy thử ví dụ này một vài lần bằng cách dùng các hàm *colormap* khác nhau và thay đổi tham số n:

```
>> colormap(jet(n))
>> n=8;
>> colormap(jet(n))
>> pcolor([1:n+1;1:n+1]')
>> title('using pcolor to display a colormap')
```

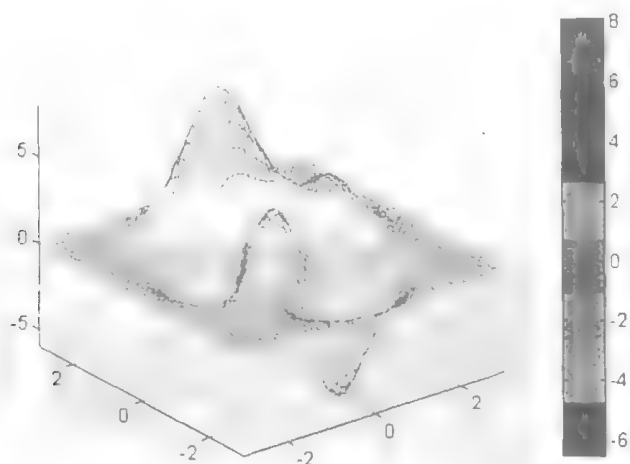


Hình 18.4

Hàm *colorbar* thêm một thanh màu đứng hoặc thanh màu ngang (cân chỉnh màu) vào cửa sổ hình vẽ của bạn, đưa ra biểu đồ màu cho trực diện tại *colorbar('h')* định vị thanh màu

ngang dưới hình vẽ hiện tại của bạn `colorbar('v')` định vị thanh màu đứng về bên phải hình vẽ của bạn `colorbar` không có đối số thì là thêm một thanh màu ngang, nếu thanh màu này không tồn tại hoặc là cập nhật nếu nó tồn tại.

```
>> [X,Y,Z] = peaks;
>> mesh(X,Y,Z);
>> colormap(hsv)
>> axis([-3 3 -3 3 -6 8])
>> colorbar
```



Hình 18.5

18.9 Thiết lập và thay đổi bảng màu

Thực tế **colormaps** là các ma trận, có nghĩa là bạn có thể thao tác chúng giống như bất kỳ một ma trận nào khác. Hàm **brighten** nhờ vào đặc điểm này thay đổi colormap độ tăng hoặc giảm độ nhạy của các màu đậm. **Brighten(n)** cùng với **brighten(-n)** phục hồi colormap ban đầu. Lệnh **newmap=brighten(n)** tạo một thanh màu sáng hơn hoặc tối hơn của colormap hiện tại mà không làm thay đổi biểu đồ màu hiện tại. Lệnh **newmap=brighten(cmap,n)** điều chỉnh phiên bản của thanh màu đã được khai báo mà không làm ảnh hưởng đến colormap hiện tại hoặc **cmap.brighten(gcf, n)** làm sáng tất cả các đối tượng trong hình vẽ hiện tại.

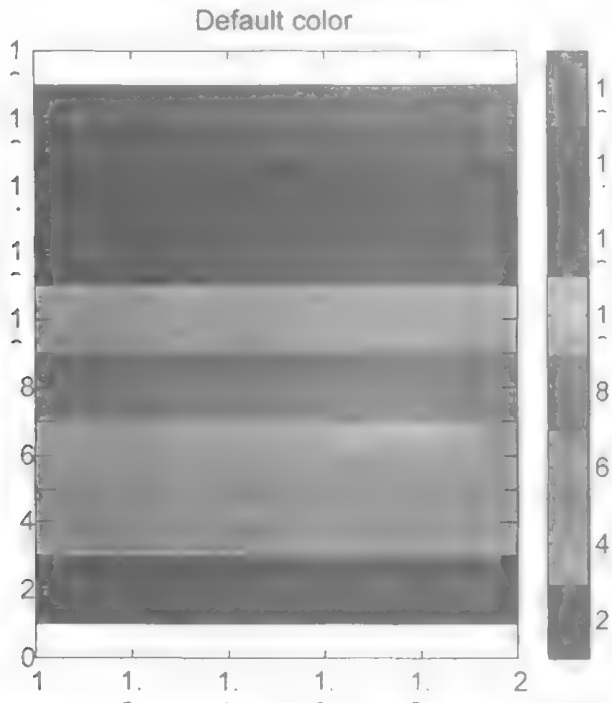
Bạn có thể tạo một colormap của riêng bạn bằng cách đưa ra một ma trận **mymap** m hàng, 3 cột và cài đặt nó cùng với **colormap(mymap)** mỗi giá trị trong một ma trận colormap phải thuộc khoảng từ 0 đến 1. Nếu bạn cố gắng sử dụng một ma trận với nhiều hơn hoặc ít hơn 3 cột hoặc chứa một giá trị nào đó bé thua 0 hoặc lớn 1 colormap sẽ đưa ra thông báo lỗi.

Bạn có thể kết nối các colormap theo kiểu toán học. Mặc dù kết quả đôi khi không thể đoán trước được. Ví dụ, biểu đồ có tên gọi là pink:

```
>> pinkmap = sqrt (2/3*gray+1/3*hot);
```

Bởi vì colormap là các ma trận, chúng có thể được vẽ đồ thị. Lệnh *rgbplot* sẽ vẽ đồ thị các giá trị của colormap tương tự như lệnh *plot* nhưng sử dụng màu đỏ, màu xanh lá cây và xanh da trời cho nét vẽ *rgbplot(gray)* cho biết cả ba màu tăng tuyến tính và đồng đều. Lệnh *rgbplot* với một số colormap khác như *jet*, *hsv* và *prism*.

Giá trị hiện tại của *cmin* và *cmax* được trả lại bằng *caxis* không có đối số. Chúng thường là những giá trị lớn nhất và nhỏ nhất của dữ liệu. *caxis([cmin cmax])* sử dụng colormap nguyên bản cho dữ liệu trong dải giữa *cmin* và *cmax*, nhưng điểm dữ liệu lớn hơn *cmax* sẽ bị chia ra thành các màu kết hợp với *cmax*. Và những điểm dữ liệu có giá trị nhỏ hơn *cmin* sẽ bị chia ra thành các màu kết hợp với *cmin*. Nếu *cmin* nhỏ hơn *min(data)* hoặc *cmax* lớn hơn *max(data)*, thì các màu kết hợp với *cmin* hoặc *cmax* sẽ không bao giờ được sử dụng chỉ một phần nhỏ của colormap được sử dụng. *caxis('auto')* sẽ khôi phục giá trị mặc định của *cmin* và *cmax*.



Hình 18.6

Ví dụ sau được minh họa trong colorplate4 .

```
>> pcolor([1:17;1:17])
```



```
>> title('Default color range')
>> colormap(hsv(8))
>> axis('auto')
>> colorbar
>> caxis
ans =
     1    17
```

MẢNG TẾ BÀO VÀ CẤU TRÚC

MATLAB 5.0 giới thiệu 2 loại dữ liệu mới có tên gọi là mảng tế bào và cấu trúc. Mảng tế bào được xem như một mảng của các số nh phân hoặc là như bộ chứa có thể lưu giữ nhiều kiểu dữ liệu khác nhau. Cấu trúc là những mảng dữ liệu hướng đối tượng xây dựng cùng với tên các trường có thể chứa nhiều kiểu dữ liệu khác nhau, bao gồm mảng tế bào và các cấu trúc khác. Cấu trúc cung cấp cho ta phương tiện thuận lợi để nhóm các kiểu dữ liệu khác nhau. Những kiểu dữ liệu mới này, mảng tế bào và cấu trúc tạo cho bạn khả năng tổ chức dữ liệu thành các gói rất thuận tiện.

19.1 Mảng tế bào

Mảng tế bào là những mảng MATLAB mà các phần tử của nó là các tế bào. Mỗi tế bào trong mảng tế bào chứa các kiểu dữ liệu của MATLAB bao gồm mảng số, văn bản, đối tượng đặc trưng, các mảng tế bào và cấu trúc. Ví dụ một tế bào của mảng tế bào có thể là mảng số, loại khác là kiểu chuỗi văn bản, loại khác là vector các giá trị số phức. Các mảng tế bào có thể được xây dựng với số chiều lớn hơn 2, tuy nhiên để cho thuận tiện khi xét người ta lấy số chiều là 2.

19.2 Xây dựng và hiển thị mảng tế bào

Mảng tế bào có thể được xây dựng bằng cách dùng câu lệnh gán, hoặc chỉ định mảng trước bằng cách sử dụng hàm tế bào sau đó gán dữ liệu cho mảng.

Như mọi loại mảng khác, mảng tế bào có thể tạo ra bằng cách gán dữ liệu cho từng tế bào độc lập ở cùng một thời điểm. Có hai cách khác nhau thêm nhập vào mảng tế bào. Nếu bạn sử dụng cú pháp mảng tiêu chuẩn, bạn phải để các tế bào trong dấu ngoặc "{ }". Ví dụ:

```
>> A(1, 1) = {[1 2 3; 4 5 6; 7 8 9]};
>> A(1, 2) = {2 + 3i};
>> A(2, 1) = {'A text string'};
```

```
>> A(2, 2) = {12: -2:0};
```

Dấu ngoặc nhọn bên phía phải của dấu bằng chỉ ra rằng biểu thức là một tế bào, hay còn gọi là chỉ số tế bào. Cách viết sau tương đương với cách viết trên:

```
>> A{1, 1} = [' 2 3: 4 5 6: 7 8 9];
```

```
>> A{1, 2} = 2+3i;
```

```
>> A{2, 1} = 'A text string ';
```

```
>> A{2, 2} = 12: -2: 0;
```

Dấu ngoặc nhọn bên trái chỉ ra rằng A là một mảng tế bào và biểu thức đặt bên trong là khai báo tế bào.

MATLAB hiển thị mảng A như sau:

```
>> A
```

```
A =
```

```
 [3X3 double] 2.0000+ 3.0000 i
```

```
 'A text string' [1x7 double]
```

Để hiển thị nội dung của mỗi tế bào trong mảng tế bào ta dùng hàm *celldisp*, hiển thị nội dung của riêng một tế bào, truy nhập vào tế bào có sử dụng dấu ngoặc nhọn. Ví dụ:

```
>> A{2,2}
```

MATLAB hiển thị sơ đồ cấu trúc đồ họa mảng tế bào trong một cửa sổ bằng việc gọi hàm *cellplot*.

Hàm *cell* làm việc với mảng tế bào bằng việc tạo ra các mảng trống theo kích cỡ của mảng. Ví dụ:

```
>> C= cell (2, 3)
```

```
C=
```

```
 [ ] [ ] [ ]
```

```
 [ ] [ ] [ ]
```

19.3 Tổ hợp và khôi phục mảng tế bào

Nếu bạn gán dữ liệu cho tế bào ngoài số chiều hiện có của mảng. MATLAB sẽ tự động mở rộng mảng và điền vào giữa ma trận số rỗng. Chú ý khái niệm “{ }” thay cho ma trận tế bào rỗng và “[]” thay cho mảng số ma trận rỗng.

Sử dụng dấu móc vuông để kết nối mảng tế bào:

```
>> C = {A B}
```

```
C =
```

```
    [3x3 double]    2.0000+ 3.0000i    [1x2 double]    'John Smith'
```

```
    'A text string'    [1x7 double]    [2.0000+3.0000i]    [ 5 ]
```

```
>> C=[A:B]
```

```
C =
```

```
    [3x3 double]    2.0000 + 3.0000 i
```

```
    'A text string'    [ 1x7    double ]
```

```
    [ 1x2    double ]    'John Smith'
```

```
    [ 2.0000+ 3.0000i]    [      5 ]
```

Một tập con các tế bào có thể được tách ra tạo thành một mảng tế bào mới. Nếu D là một mảng tế bào 3x3, người ta có thể tách ra để tạo thành một mảng tế bào mới 2x2 như sau:

```
>> F = D(2:2,2:3);
```

Hàm **reshape** có thể được sử dụng để thay đổi cấu hình của một mảng tế bào nhưng không thể dùng để thêm vào hoặc bớt đi tế bào.

```
>> X = cells(3,4);
```

```
>> size(X)
```

```
ans =
```

```
     3     4
```

```
>> X= reshape(X,6,2);
```

```
>> size(Y)
```

```
ans =
```

```
     6     2
```

19.4 Truy nhập vào trong mảng tế bào

Để truy nhập dữ liệu chứa trong các phần tử của mảng tế bào, sử dụng dấu ngoặc nhọn. Dùng dấu ngoặc đơn thâm nhập một phần tử như là một tế bào. Để truy nhập nội dung của phần tử trong mảng tế bào, kết nối các biểu thức như sau:

```
>> x = B{2,2}           % truy nhập nội dung của tế bào.
```

```

x =

         5

>> class(x)

ans =

      double

>> y = B[2,2]      % truy nhập vào bản thân tế bào.

y =

         [5]

>> class(y)

ans =

      cell

>> B{1,1} (1,2) % truy nhập vào phần tử thứ hai của
                % vectơ trong tế bào

ans =

         2

Để truy nhập dải các phần tử trong mảng tế bào, sử dụng hàm deal

>> [a,b] = deal(B{2,:})

a =

      2.0000+ 3.0000i

b =

         5

```

Hàm *deal* cần một danh sách các biến phân biệt nhau bởi dấu phẩy. Biểu thức $B\{2,: \}$ có thể sử dụng ở mọi nơi và dấu phẩy dùng để phân tách danh sách các biến. Do đó, $B\{2,: \}$ tương đương với $B(2,1)$ và $B(2,2)$

19.5 Mảng tế bào của chuỗi kí tự

Một trong những ứng dụng phổ biến của mảng tế bào là xây dựng một mảng văn bản. Mảng chuỗi kí tự tiêu chuẩn đòi hỏi tất cả các chuỗi đều có chung độ dài. Bởi vì mảng tế bào có thể chứa nhiều kiểu dữ liệu khác nhau trong mỗi phần tử, chuỗi kí tự trong mảng tế bào không có giới hạn này. Ví dụ:

```
>> T = {' Tom' ' Disk'}
```

```
T=
```

```
 'Tom'
```

```
 'Disk'
```

19.6 Cấu trúc

Cấu trúc là những đối tượng MATLAB có tên “bộ chứa dữ liệu” còn gọi là *fields*. Như mọi phần tử của mảng tế bào, trường cấu trúc có thể có bất cứ một kiểu dữ liệu nào. Chúng khác ở chỗ cấu trúc trường được truy nhập bằng tên phổ biến hơn là chỉ số, và không có sự hạn chế nào về chỉ số cũng như cấu hình của các trường cấu trúc. Cũng giống như mảng tế bào, cấu trúc có thể được nhóm lại với nhau tạo thành mảng và mảng tế bào. Một cấu trúc đơn là một mảng cấu trúc 1x1.

19.7 Xây dựng mảng cấu trúc

Cấu trúc sử dụng dấu “.” để truy nhập vào trường. Xây dựng một cấu trúc đơn giản như gán dữ liệu vào các trường độc lập. Ví dụ sau tạo một bản ghi *client* cho thư viện kiểm tra.

```
>> client.name = ' John Doe';
```

```
>> client.cost = 86.50;
```

```
>> client.test.AIC = [6.3 6.8 7.1 7.0 6.7 6.5 6.3 6.4]
```

```
>> client.test.CHC = [2.8 3.4 3.6 4.1 3.5];
```

```
>> client
```

```
client =
```

```
    name: ' John Doe '
```

```
    cost:86.50
```

```
    test: [1x1 struct]
```

```
>> client.test
```

```
ans=
```

```
    AIC:6.3000 6.8000 7.1000 7.0000 6.7000 6.5000 6.3000 6.4000
```

```
    CHC:2.8000 3.4000 3.6000 4.1000 3.5000
```

Bây giờ tạo bản ghi *client* thứ hai:

```
>> client(2).name = ' Alice Smith ';
>> client(2).cost = 112.35;
>> client(2).test.AIC = [5.3 5.8 7.0 6.5 6.7 5.5 6.0 5.9]
>> client(2).test.CHC = [3.8 6.3 3.2 3.1 2.5]
>> client
client =
1x2 struct array with field
    name
    cost
    test
```

Cấu trúc cũng có thể được xây dựng bằng cách dùng hàm **struct** để tạo trước một mảng cấu trúc. Cú pháp là: ('field1', V1, 'field2', V2, ...) trong đó field1, field2, .v.v.. là các trường, và các mảng V1, V2, v.v.... phải là các mảng tế bào có cùng kích thước, cùng số tế bào, hoặc giá trị. Ví dụ, một mảng cấu trúc có thể được tạo ra như sau:

```
>> N = {' John Doe ', ' Alice Smith'};
>> C = {86.50, 112.35};
>> P = {[10.00 20.00 45.00]};
>> bills = struct('name',N,'cost',C,'payment',P)
bills=
1x2 struct array with fields
    name
    cost
    payment
```

19.8 Truy nhập vào các trường cấu trúc

Bởi vì nội dung cấu trúc là tên nhiều hơn là chỉ số, như trong trường hợp mảng tế bào, tên của các trường trong cấu trúc phải được biết đến để truy nhập dữ liệu chứa trong chúng. Tên của các trường có thể được tìm thấy ở trong ở trong cửa sổ lệnh đơn giản là chỉ việc nhập vào tên của cấu trúc. Tuy nhiên ở trong M-file, một hàm cần thiết được tạo ra để cập nhật các tên trường đó. Hàm **fieldname** trả lại một mảng tế bào có chứa tên của các trường trong một cấu trúc.

```
>> T = fieldnames(bills)
```

```
T =
```

```
    'name '
```

```
    'cost '
```

```
    'payment '
```

Có hai phương pháp để truy nhập vào trường cấu trúc. Chỉ số trực tiếp sử dụng kĩ thuật chỉ mục thích hợp, như phương pháp truy nhập trường cấu trúc, và chỉ số mảng thích hợp để truy nhập vào một số hoặc một mảng tế bào. Sau đây là một ví dụ dựa trên cấu trúc **bills** và **client** đã xét ở trên:

```
>> bills.name
```

```
ans =
```

```
    John Doe
```

```
ans =
```

```
    Alice Smith
```

```
>> bills(2).cost
```

```
ans =
```

```
    112.3500
```

```
>> bills(1)
```

```
ans =
```

```
    name: ' John Doe '
```

```
    cost: ' 86.5000 '
```

```
    payment: 10.000  20.0000  45.0000
```

```
>> baldue = bills(1).cost - sum(bills(1).payment)
```

```
baldue =
```

```
    6.5000
```

```
>> bills(2).payment(2)
```

```
ans =
```

```
    12.3500
```

```
>> client(2).test AIC(3)
```



```
ans=
```

```
7.000
```

Phương pháp chỉ mục trực tiếp thường được sử dụng để truy nhập giá trị trường. Tuy nhiên, ở các M-file nếu tên các trường được gọi ra từ hàm *fieldnames*, thì hàm *getfield* và *setfield* có thể được sử dụng để truy nhập dữ liệu trong cấu trúc. Ví dụ:

```
>> getfield(bills,{1} 'name')    % tương tự như bills(1).name
```

```
ans=
```

```
John Doe
```

```
>> T = fieldnames(bills);
```

```
>> getfield(bills,{2},T{3},{2})%tương tự như s(2).payment(2)
```

```
ans=
```

```
12.3500
```

Ví dụ sau trả lại cấu trúc có chứa cùng kiểu dữ liệu như cấu trúc nguyên thủy với một giá trị bị thay đổi. Dòng lệnh tương đương của `client(2).test.AIC(3) = 7.1` là:

```
>> client = setfield(client,{2},'test','AIC',{3},7.1)
```

```
client=
```

```
1x2 struct array with fields
```

```
name
```

```
cost
```

```
test
```

```
>> client(2).test.AIC(3)
```

```
ans=
```

```
7.1000
```

Một trường có thể được thêm vào trong một mảng cấu trúc chỉ đơn giản bằng cách gán giá trị cho trường cấu trúc mới.

```
>> client(1).addr = {' MyStreet',' MyCity '}
```

```
client =
```

```
1x2 struct array with fields
```

```
name
```

cost

test

addr

Một trường có thể được bỏ đi khỏi cấu trúc (hoặc một mảng cấu trúc) bằng lệnh *rmfield*. *S=rmfield(S, 'field')* sẽ bỏ đi trường *field* từ cấu trúc *S*. *S=rmfield(S, F)*, trong đó *F* là một mảng tế bào của tên các trường, bỏ đi nhiều hơn một trường từ cấu trúc *S* tại một thời điểm.

```
>> client = rmfield(client,'addr')
```

```
client =
```

```
1x2 struct array with fields
```

```
name
```

```
cost
```

```
test
```

19.9 Sự nghịch đảo và hàm kiểm tra

Sự nghịch đảo giữa các mảng tế bào và các cấu trúc bằng cách dùng hàm *struct2cell* và *cell2struct*. Tên trường phải được cung cấp đầy đủ cho *cell2struct* và bị mất đi khi chuyển thành một mảng tế bào từ một cấu trúc. Sự chuyển đổi từ mảng số và mảng xâu kí tự thành mảng tế bào bằng cách sử dụng hàm *num2cell* và *cellstr*. Ngược lại chuyển đổi từ một mảng tế bào thành mảng kí tự bằng hàm *char*.

Mặc dù hàm *class* trả về kiểu kiểu dữ liệu của đối tượng, *class* vẫn không thuận tiện sử dụng để kiểm tra kiểu dữ liệu. Hàm *isax*, ('class') trả lại *true* nếu *x* là một đối tượng kiểu 'class'. Ví dụ, *isa*(client, 'struct') sẽ trả lại *true*. Để thuận tiện, một số hàm kiểm tra số khác có sẵn trong thư viện chương trình như: *isstruct*, *iscell*, *ischar*, *isnumeric*, và *islogical*.

BIỂU TƯỢNG CỦA HỘP CÔNG CỤ TOÁN HỌC

Hộp công cụ toán học là một tập hợp các công cụ (hàm) để MATLAB giải các bài toán. Có các công cụ để tổ hợp, đơn giản hoá, tích phân, vi phân và giải các phép toán đại số và phép toán vi phân. Các công cụ khác sử dụng trong đại số học tuyến tính để chuyển đổi chính xác dạng nghịch đảo, định thức và các khuôn mẫu tiêu chuẩn.

Các công cụ trong **Symbolic Math Toolbox** được tạo nên từ chương trình phần mềm mạnh có tên là Maple® phát triển khởi đầu từ trường đại học Waterloo ở Ontario, Canada và bây giờ là phần mềm của hãng Waterloo Maple Software. Khi bạn yêu cầu MATLAB thực hiện một phép toán, nó sẽ sử dụng các hàm của **Symbolic Math Toolbox** để làm việc này và trả lại kết quả ở cửa sổ lệnh.

20.1 Biểu thức và các đối tượng đặc trưng

MATLAB cơ sở sử dụng một số các kiểu đối tượng khác nhau để lưu trữ giá trị. Biến số học dùng để lưu trữ giá trị số học, ví dụ như $x=2$, mảng kí tự để lưu trữ chuỗi văn bản, ví như: $t = 'A \text{ text string}'$. Hộp công cụ toán học đặc trưng dùng những đối tượng toán học thay thế các biến và các toán tử, ví dụ: $x = \text{sym}('x')$. Các đối tượng toán học được sử dụng bởi MATLAB trong nhiều trường hợp tương tự như các biến số học và chuỗi được sử dụng. Biểu thức toán học là những biểu thức có chứa đối tượng toán học thay thế cho các số, hàm, toán tử và các biến. Các biến không yêu cầu phải định nghĩa trước. Thuật toán là công cụ thực hành để giải quyết những bài toán trên cơ sở biết được những quy luật và sự nhận dạng các biểu tượng được đưa ra, chính xác như cái cách bạn giải bằng đại số học và sự tính toán... Các ma trận toán học là những mảng mà phần tử của nó là các đối tượng toán học hoặc các biểu thức.

20.2 Tạo và sử dụng các đối tượng đặc trưng

Đối tượng đặc trưng được xây dựng từ những chuỗi kí tự hoặc các biến số học sử dụng hàm **sym**. Ví dụ $x = \text{sym}('x')$ tạo ra một biến đặc trưng x , $y = \text{sym}('y')$ tạo ra một biến đặc trưng y , $y = \text{sym}('1/3')$ tạo ra một biến đặc trưng y mang giá trị $1/3$. Giả sử biến đặc trưng được định nghĩa, nó có thể được sử dụng trong các biểu thức toán học tương tự như các biến số học được sử dụng trong MATLAB. Nếu như các biến x , y được tạo ra trước đó thì lệnh $z = (x+y)/(x-2)$ sẽ tạo một biến mới z bởi vì biểu thức mà nó thay thế có mang một hay nhiều biến đặc trưng x hoặc y .

Một đối tượng số học có thể chuyển thành đối tượng đặc trưng. Dưới đây là một ví dụ:

```
>> m = magic(3)           % tạo một ma trận số  
  
m =  
  
     8     1     6  
     3     5     7  
     4     9     2  
  
>> M = sym(m)             % tạo một ma trận đặc trưng từ m  
  
M =  
  
 [ 8, 1, 6]  
 [ 3, 5, 7]  
 [ 4, 9, 2]  
  
>> det(M)                 % xác định định thức của ma trận đặc trưng M  
  
ans =  
  
    -360
```

Ví dụ này xây dựng một ma trận vuông 3×3 , chuyển đổi thành ma trận đặc trưng, và tìm định thức của ma trận.

Hàm **sym** cho phép bạn lựa chọn định dạng cho sự hiển thị đặc trưng của giá trị số. Cú pháp là: $S = \text{sym}(A, \text{fmt})$ trong đó A là giá trị số hoặc ma trận còn fmt là một đặc tính định dạng tùy chọn, có thể là 'f', 'r', 'e', hoặc 'd'. Giá trị mặc định là 'r'. Nếu chọn 'f' tương ứng hệ chữ số lục phân, 'r' tương ứng chữ số hữu tỉ, 'e' tương tự như 'r' nhưng ở dạng chính tắc hàm mũ, còn 'd' tương ứng chữ số hệ thập phân.

Dưới đây là một số ví dụ về sự hiển thị của một số định dạng tùy chọn:

Biểu thức tượng trưng	Sự trình bày trong MATLAB
1	<code>x=sym('x')</code>
$2x^3$	
$y = \frac{1}{\sqrt{2x}}$	<code>x=sym('x')</code>
$\cos(x^2) - \sin(2x)$	<code>x=sym('x')</code>
$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$	<code>syms('a','b','c','d');</code>
$f = \int_a^b \frac{x^3}{\sqrt{1-x}} dx$	<code>syms x a b</code>
	<code>f = int (x^3/ sqrt (1-x),a, b)</code>

Các hàm đặc trưng của MATLAB cho phép bạn thao tác những biểu thức này theo nhiều cách khác nhau. Ví dụ:

```
>> x = sym('x')      % tạo một biến đặc trưng x
>> diff(cos(x))      % đối của cos(x) với biến số là x
ans =
    -sin(x)
>> sym('a','b','c','d') % tạo biến số đặc trưng a, b, c và d
>> M = [a, b, c, d]   % tạo một ma trận đặc trưng
M =
    [a, b]
    [c, d]
>> det(M)             % tìm định thức của ma trận đặc trưng M
ans =
    a*b - b*c
```

Trong ví dụ đầu tiên, x được định nghĩa như một biến đặc trưng trước khi nó được sử dụng trong biểu thức, tương tự như vậy biến số phải được gán một giá trị trước khi chúng được sử

dùng. Điều này cho phép MATLAB xem xét $\cos(x)$ như một biểu thức đặc trưng, và do vậy $\text{diff}(\cos(x))$ là một phép toán đặc trưng hơn là một phép toán số học. Trong ví dụ số 2, hàm **syms** thường được định nghĩa là một số biến số đặc trưng. **syms**('a', 'b') tương đương với $a = \text{sym}('a');$ $b = \text{sym}('b');$. MATLAB biết rằng $M=[a, b; c, d]$ là một ma trận đặc trưng bởi vì nó chứa đựng một biến số đặc trưng, và do đó $\text{det}(M)$ là một phép toán đặc trưng.

Trong MATLAB, câu lệnh **funcarg** tương đương với **func('arg')**, trong đó **func** là một hàm, còn **arg** là một chuỗi đối số kí tự. MATLAB phân biệt **syms a b c d** và **syms**('a', 'b', 'c', 'd') là tương đương nhưng như các bạn biết công thức đầu tiên dễ thực hiện hơn.

Chúng ta xem xét kỹ hơn ví dụ thứ hai đã nêu ở trên:

```
>> a = 1; b = 2; c = 3; d = 4      % định nghĩa biến số a đến d
```

```
>> M = [a,b;c,d]                  % M là một ma trận số
```

```
M =
```

```
     1     2
```

```
     3     4
```

```
>> size(M)                        %M là một ma trận bậc hai
```

```
ans =
```

```
     2     2
```

```
>> class(M)                       % Có những loại đối tượng nào là M?
```

```
ans =
```

```
double
```

```
>> M = ['a, b; c, d']              % M là một chuỗi đặc trưng
```

```
M =
```

```
    [a, b; c, d]
```

```
>> size(M)                        % M là một vector hàng của 9 kí tự
```

```
ans =
```

```
     1     9
```

```
>> class(M)
```

```
ans =
```

```
char
```

```

>> M = sym('[a,b;c,d]')           % một đối tượng đặc trưng nhưng
                                   % không phải là một ma trận

M=
    [a,b;c,d]

>> size(M)                         % M là một vector 3 phần tử (2 dấu phẩy)
ans =
     1     3

>> class(M)
ans =
    sym

>> syms a b c d                   % định nghĩa biến số đặc trưng a đến d

>> M = [a,b;c,d]                  % M là một ma trận đặc trưng

M=
    [a, b]
    [c, d]

>> size(M)
ans =
     2     2

>> class(M)
ans =
    sym

>> a = 1; b = 2; syms c d         % định nghĩa một biến cố định từ a

>> M = [a,b;c,d]                  % M là một ma trận đặc trưng từ a đến d

M=
    [1, 2]
    [c, d]

>> size(M)

```


ans=

sym

Trong ví dụ này, M được định nghĩa theo 5 cách:

- Kiểu thứ nhất: nó gần giống với ma trận bậc hai.
- Kiểu thứ hai là một chuỗi kí tự.
- Kiểu thứ ba là một đối tượng đặc trưng hợp lệ, nhưng nó không thể sử dụng trong mọi trường hợp

- Kiểu thứ tư là một ma trận bậc hai.
- Kiểu cuối cùng cho thấy biến số là biến đặc trưng có kết hợp trong biểu thức đặc trưng để tạo thành ma trận đặc trưng.

Biểu thức đặc trưng không có biến được gọi là hàm đặc trưng. Khi hàm đặc trưng hiển thị, chúng đôi khi khó mà phân biệt được với số nguyên. Ví dụ:

```
>> f=sym(3)           %tạo một hằng đặc trưng
```

f=

3

```
>> class(f)           % kiểu của đối tượng f là gì
```

ans=

sym

```
>> g = sym(pi)
```

g=

pi

```
>> class(g)
```

ans=

sym

```
>> h = sym(sin(pi/4))
```

h=

sqrt(1/2)

```
>> class(h)
```

ans=

sym

20.4 Biến đặc trưng

Khi làm việc với biểu thức đặc trưng có nhiều hơn một biến đặc trưng, chính xác hơn một biến là biến độc lập. Nếu MATLAB không được chỉ ra đâu là biến độc lập thì nó sẽ nhận biến nào gần x nhất theo thứ tự chữ cái

Biến độc lập đôi khi còn được gọi là biến tự do. Bạn có thể yêu cầu MATLAB chỉ ra biến nào trong biểu thức đặc trưng. Để biết được ta sử dụng hàm *findsym*.

```
>> syms a s t u omega i j      % định nghĩa các biến đặc trưng
```

```
>> findsym(a*t+s/(u+3),1)      % u là gần x nhất
```

```
ans =
```

```
u
```

```
>> findsym(sin(a+omega),1)      % omega gần x nhất
```

```
ans =
```

```
omega
```

```
>> findsym(3*i + 4*j)           % i và j tương tự như sqrt(-1)
```

```
ans =
```

```
i
```

Nếu *findsym* không tìm thấy biến đặc trưng, nó sẽ trả lại chuỗi rỗng.

20.5 Phép toán trên biểu thức đặc trưng

Giả sử bạn đã tạo được biểu thức đặc trưng, bạn rất có thể muốn thay đổi nó bằng bất cứ cách nào. Bạn muốn lấy ra một phần của biểu thức, kết hợp hai biểu thức hoặc tìm một giá trị số của một biểu thức đặc trưng. Có rất nhiều công cụ cho phép bạn làm điều này.

Tất cả các hàm đặc trưng, (với vài điểm đặc biệt sẽ nói ở phần sau) dựa trên các biểu thức đặc trưng và các mảng đặc trưng. Kết quả giống như một số nhưng nó là một biểu thức đặc trưng. Như chúng ta đã nói ở trên, bạn có thể tìm ra đâu là kiểu số nguyên, một chuỗi đặc trưng hoặc một đối tượng đặc trưng bằng cách sử dụng hàm *class* từ MATLAB cơ sở.

20.6 Tách các tử số và mẫu số

Nếu biểu thức của bạn là một đa thức hữu tỉ hoặc có thể mở rộng tới một đa thức hữu tỉ tương đương (bao gồm toàn bộ các phần tử của tử số có chung mẫu số), bạn có thể tách tử số và mẫu số bằng cách sử dụng hàm *numden*. Ví dụ:

$$m = x^2, f = a x^2/(b-x) \quad g = 3 x^2/2 + 2 x/3 - 3/5.$$

$$n = (x^2 + 3)/(2x - 1) + 3x/(x-1)$$

numden tổ hợp hoặc hữu tỉ hoá biểu thức nếu cần thiết, và trả lại kết quả tử số và mẫu số. Câu lệnh MATLAB được thực hiện như sau:

```
>> sym x a b           % tạo một số biến đặc trưng
```

```
>> m = x^2             % tạo một biểu thức đơn giản
```

```
m =
```

$$x^2$$

```
>> [n,d] = numden(m)    % tách tử số và mẫu số.
```

```
n =
```

$$x^2$$

```
d =
```

$$1$$

```
>> f = a*x^2/(b-x)     % tạo một biểu thức liên quan
```

```
f =
```

$$a*x^2/(b-x)$$

```
>> [n d] = numden(f)    % tách tử số và mẫu số.
```

```
m =
```

$$-a*x^2$$

```
d =
```

$$-b + x$$

Hai biểu thức đầu tiên cho ta kết quả như mong muốn

```
>> g = 3/2*x^2 + 2*x - 3/4 % tạo một biểu thức khác.
```

```
g =
```

$$3/2*x^2 + 2*x - 3/4$$

```
>> [n,d] = numden(g)     % hữu tỉ hoá và tách các phần
```

```
n =
```

$$6*x^2 + 8*x - 3$$

```
d =
```

$$4$$

```
>> h = (x^2 + 3)/(2*x - 1) + 3*x/(x - 1) % tổng của đa thức hữu tỉ
```

```
h =
```

$$x^3 + 5x^2 - 3$$

```
d = (2*x - 1)*(x - 1)
```

```
>> h2 = n/d % tạo lại biểu thức cho h
```

```
h2 =
```

$$(x^3 + 5x^2 + 3)/(2x - 1)$$

Hai biểu thức g và h được hữu tỉ hoá hoặc trở về biểu thức đơn giản với một tử số và mẫu số, trước khi các phần tử được tách có thể chia tử số cho mẫu số tạo lại biểu thức nguyên gốc.

20.7 Phép toán đại số tiêu chuẩn

Một số phép toán tiêu chuẩn có thể biểu diễn trên biểu thức đặc trưng sử dụng các toán tử quen thuộc. Ví dụ cho hai hàm:

$$f = 2x^2 + 3x - 5 \quad g = x^2 - x + 7$$

```
>> sym('x') % định nghĩa một biến số đặc trưng
```

```
>> f = (2*x^2 + 3*x - 5) % định nghĩa biểu thức đặc trưng f và g
```

```
f =
```

$$(2x^2 + 3x - 5)$$

```
>> x^2 - x + 7
```

```
g =
```

$$x^2 - x + 7$$

```
>> f +
```

```
ans =
```

$$3x^2 + 2x + 2$$

```
>> f - g % tìm biểu thức của f-g
```

```
ans =
```

$$x^2 + 4x - 12$$

```
>> f*g % tìm một biểu thức của f*g
```

```
ans =
```

$$(2x^2 + 3x - 5)(x^2 - x + 7)$$

```
>> f/g          % tìm một biểu thức của f/g
ans =
      (2*x^2 + 3*x - 5)/(x^2 - x + 7)
>> f^(3*x)      % tìm một biểu thức cho f^x
ans =
      (2*x^2 + 3*x - 5)*3*x
```

Thực sự là một phép toán trên bất cứ biểu thức nào chứa ít nhất một biến số đặc trưng sẽ cho kết quả của một biểu thức đặc trưng. bạn hãy tổ hợp các biểu thức cố định để tạo những biểu thức mới. Ví dụ:

```
>> a = 1; b = 3/2; x = sym('x'); % tạo một số và những biến số đặc trưng
>> f = sin(a - x)    % tạo một số biểu thức
ans=
      -sin(x-1)
>> g = sin(b*x^2)
ans=
      sin(3/2*x^2)
>> b*f/(g - 5)+ x    % kết hợp chúng
ans =
      -3/2*sin(x - 1)/(sin(3/2*x^2)- 5)+ x
```

Tất cả các phép toán này đều thực hiện tốt với các đối số là mảng.

20.8 Các phép toán nâng cao

MATLAB có thể biểu diễn nhiều phép toán nâng cao hơn biểu thức đặc trưng. Hàm *compose* kết hợp $f(x)$ và $g(x)$ thành $f(g(x))$. Hàm *finverse* tìm hàm nghịch đảo của một biểu thức và hàm *symsum* tìm tổng đặc trưng của một biểu thức. Ví dụ:

```
f = 1/(1 + x^2)  g = sin(x)    h = x/(1 + u^2)  k = cos(x+v)
>> syms x u v    % định nghĩa 3 biến đặc trưng
>> f = 1/(1+x^2)  % tạo 4 biểu thức
>> g = sin(x)
```

```
>> h = x/(1 + u^2)
>> k = cos(x + v)
>> compose(f,g)    % tìm biểu thức của f(g(x))
```

```
ans =
```

```
sym(1/(1 + x^2))
```

compose có thể được sử dụng ở các hàm mà có các biến độc lập khác nhau.

```
>> compose(h,k)    % cho h(x), k(x), tìm h(k(x))
```

```
ans =
```

```
cos(x + v)/(1 + u^2)
```

```
>> compose(h,k,u,v)    % cho h(u), k(v), tìm h(k(v))
```

```
ans =
```

```
x/(1 + cos(2*v)^2)
```

Hàm nghịch đảo của một biểu thức, gọi là $f(x)$, là biểu thức $g(x)$ mà thoả mãn điều kiện $g(f(x)) = x$. Ví dụ hàm nghịch đảo của e^x là $\ln(x)$, do vậy $\ln(e^x) = x$. Hàm nghịch đảo của $\sin(x)$ là $\arcsin(x)$, và hàm nghịch đảo của $1/\tan(x)$ là $\arctan(1/x)$. Hàm **finverse** trở thành hàm nghịch đảo của một biểu thức. Chú ý **finverse** trả lại duy nhất một kết quả, thậm chí nếu kết quả đó không là duy nhất.

```
>> syms x a b c d z    % định nghĩa một số biến đặc trưng
```

```
>> finverse(1/x)    % nghịch đảo của 1/x là x
```

```
ans =
```

```
1/x
```

```
>> finverse(x^2)    % tìm một trong các giải pháp để  $g(x^2) = x$ 
```

```
ans =
```

```
x^(1/2)    -
```

```
>> finverse(a*x + b) % tìm giải pháp để  $g(f(x)) = x$ 
```

```
ans =
```

```
-(b - x)/a
```

```
>> finverse(a*b + c*d - a*z,a) tìm giải pháp để  $g(f(a)) = a$ 
```

```
ans =
```

```
-(c*d - a)/(b - z)
```

Hàm **symsum** tìm tổng đặc trưng của một biểu thức. Có 4 cú pháp của hàm: **symsum(f)** trả lại tổng $\sum_0^{x-1} f(x)$, **symsum(f,s)** trả lại tổng $\sum_0^{s-1} f(s)$, **symsum(f,a,b)** trả lại tổng $\sum_a^b f(x)$, còn hàm **symsum(f, a, b, s)** trả lại tổng $\sum_a^b f(s)$.

Chúng ta cùng xem xét tổng $\sum_0^{x-1} x^2$, trả lại $x^3/3 - x^2/2 + x/6$

```
>> syms x n
>> symsum(x^2)
ans =
1/3*x^3 - 1/2*x^2 + 1/6*x
```

20.9 Hàm nghịch đảo

Mục này trình bày các công cụ để chuyển đổi biểu thức đặc trưng sang giá trị số và ngược lại. Có một số rất ít các hàm đặc trưng có thể trở thành giá trị số.

Hàm **sym** có thể chuyển đổi một chuỗi hoặc một mảng số thành sự biểu diễn đặc trưng; hàm **double** thực hiện ngược lại. **double** chuyển đổi một hằng đặc trưng (một biểu thức đặc trưng không có biến) thành giá trị số có kiểu xác định **double**.

```
>> phi = sym('(1 + sqrt(5))/2')
phi =
(1 + sqrt(5))/2
>> double(phi)      % nghịch đảo của giá trị số
ans =
1.6180
```

Hai cách trên cho ta cùng một kết quả.

Bạn đã làm việc với đa thức trên MATLAB cơ bản, sử dụng vector mà các phần tử của nó là các hệ số của đa thức. Hàm đặc trưng **sym2poli** chuyển đổi một đa thức đặc trưng thành vector của hệ số đó. Hàm **poli2sym** thì làm ngược lại, và bạn hãy khai báo biến để sử dụng trong phép toán cuối cùng.

```
>> x = sym('x')
>> f = x^3 + 2*x^2 - 3*x + 5      % f là đa thức đặc trưng
f =
```

```


$$x^3 + 2x^2 - 3x + 5$$

>> n = sym2poli(f)           % tách vector các hệ số
n =
    1    2   -3    5
>> poly2sym(n)               % tạo lại đa thức của x (mặc định)
ans =

$$x^3 + 2x^2 - 3x + 5$$

>> s = sym('s')             % định nghĩa s như là biến đặc trưng
>> poly2sym(n,s)             % tạo lại đa thức của f
ans=

$$s^3 + 2s^2 - 3s + 5$$


```

20.10 Sự thay thế biến số

Giả sử bạn có một biểu thức đặc trưng của x , và bạn muốn đổi biến thành y . MATLAB cung cấp cho bạn công cụ để thay đổi trong biểu thức đặc trưng, gọi là **subs**. Cú pháp là:

subs(f, old, new), trong đó f là một biểu thức đặc trưng, old là biến hoặc biểu thức đặc trưng, và new là biến đặc trưng, biểu thức hoặc ma trận hoặc một giá trị số hoặc ma trận. Nội dung của new sẽ thay thế old trong biểu thức f . Dưới đây là một số ví dụ:

```

>> syms a alpha b c s x     % định nghĩa một vài biến đặc trưng
>> f = a*x^2 + b*x + c       % tạo một hàm f(x)
f =

$$a*x^2 + b*x + c$$

>> subs(f,x,s)              % thay thế x bằng s trong biểu thức của f
ans=

$$a*s^2 + b*s + c$$

>> subs(f,a,[alpha;s])      % thay thế a bằng ma trận đặc trưng a
ans=

$$\{ \alpha x^2 + b x + c \}$$


$$\{ s x^2 + b x + c \}$$


```



```

>> g = 3*x^2 + 5*x - 4           % tạo một hàm khác
g =
      3*x^2 + 5*x - 4
>> h = subs(g,x,2)               % new là một giá trị số
h =
      18
>> class(h)                      % biểu diễn kết quả đó là một nội dung đặc trưng
ans =
      sym

```

Ví dụ trước biểu diễn cách ***subs*** tạo hệ số, và sau đó làm đơn giản hoá biểu thức. Từ đó kết quả của hệ số là một nội dung đặc trưng, MATLAB có thể rút gọn nó thành một giá trị đơn. Chú ý rằng ***subs*** là một hàm đặc trưng, nó trở thành một biểu thức đặc trưng, một nội dung đặc trưng thậm chí nó là một số. Để nhận một số chúng ta cần sử dụng hàm ***double*** để chuyển đổi chuỗi.

```

>> double(h)                    % chuyển đổi một biểu thức đặc trưng thành một số
ans =
      18
>> class(ans)                  % biểu diễn kết quả đó là một giá trị số
ans =
      double

```

20.11 Phép lấy vi phân

Phép lấy vi phân của một biểu thức đặc trưng sử dụng hàm ***diff*** theo một trong bốn mẫu sau đây:

```

>> syms a b c d x s           % định nghĩa một vài biến đặc trưng
>> f = a*x^3 + x^2 - b*x - c   % định nghĩa một biểu thức đặc trưng
f =
      a*x^3 + x^2 - b*x - c
>> diff(f)                    % lấy vi phân của f với x là biến mặc định
ans =

```

```

3*a*x^2 + 2*x - b
>> diff(f,a)      % lấy vi phân của f với a thay cho x
ans =
      x^3
>> diff(f,2)      % lấy vi phân f hai lần với ?
ans=
      6*a*x + 2
>> diff(f,a,2)    % vi phân 2 lần với ?
ans=
      0

```

Hàm *diff* cũng có thể thao tác trên mảng. Nếu *f* là một vector đặc trưng hoặc ma trận, *diff(f)* lấy vi phân mỗi phần tử trong mảng:

```

>> f = [a*x,b*x^2;c*x^3,d*s]      % tạo một mảng đặc trưng
f =
      [ a*x      b*x^2 ]
      [ c*x^3      d*s  ]

```

Chú ý rằng hàm *diff* cũng sử dụng trong MATLAB cơ bản để tính phép vi phân số học của một vector số và ma trận.

20.12 Phép tích phân

Hàm tích phân *int(f)* trong đó *f* là biểu thức tượng trưng, sẽ tìm ra một biểu thức tượng trưng *F* khác sao cho *diff(F)=f*. Như bạn thấy trong phần nghiên cứu phép tính, phép tích phân phức tạp hơn phép vi phân. Tích phân hoặc đạo hàm không tồn tại dưới một hình dạng khép kín; hoặc nó có thể tồn tại nhưng phần mềm không tìm ra nó hoặc phần mềm có thể tìm ra nó nhưng không đủ bộ nhớ hoặc thời gian để chạy. Khi MATLAB không tìm thấy phép tính đạo hàm nó đưa ra cảnh báo và sự thay thế tượng trưng phép tích phân đó không thể sử dụng với hàm *pretty*.

```

>> x = sym('x');
>> p = int(log(x)/exp(x^2))      % lấy tích phân
Warning: Explicit integral could not be found.

```

In C:\MATLAB\toolbox\symbolic\@sym\int.m at line 58

```
p = int(....
```

```
>> pretty(p)
```

```
ans =
```

output from pretty

Hàm tích phân, cũng như hàm vi phân đều có nhiều hơn một cú pháp. *int(f)* sẽ tìm một phép tính tích phân theo các biến độc lập mặc định, còn *int(f, s)* tìm phép lấy tích phân theo biến đặc trưng s. Khuôn mẫu *int(f, a, b)* và *int(f, s, a, b)*, trong đó a, b là các biến số, tìm ra biểu thức đặc trưng cho phép lấy tích phân theo cận từ a đến b. Tương tự cho hàm *int(f, m, n)* và

int(f, s, m, n).

```
>> syms x s m n % định nghĩa một số biến
```

```
>> f = sin(s + 2*x) % tạo một hàm tượng trưng
```

```
f =
```

$\sin(s+2x)$

```
>> int(f) % phép lấy tích phân theo biến x
```

```
ans =
```

$-1/2 \cos(s+2x)$

```
>> int(f,s) % phép lấy tích phân theo đối số s
```

```
ans =
```

$-\cos(s + 2x)$

```
>> int(f,pi/2,pi) % lấy tích phân theo biến x với cận từ pi/2 đến pi
```

```
ans =
```

$-\cos(s)$

```
>> int(f,s,pi/2,pi) % lấy tích phân theo s, cận từ pi/2 đến pi
```

```
ans =
```

$2 \cos(x)^2 - 1 - 2 \sin(x) \cos(x)$

```
>> g = simple(int(f,m,n)) % lấy tích phân theo x, cận từ m đến n
```

```
g =
```

$-1/2 \cos(s + 2n) + 1/2 \cos(s + 2m)$

Trong ví dụ này, hàm *simple* được sử dụng để đơn giản hoá kết quả của phép lấy tích phân. Chúng ta sẽ nghiên cứu thêm về hàm *simple* sau này.

Cũng như hàm *diff*, hàm lấy tích phân *int* trên mỗi phần tử của mảng đặc trưng:

```
>> syms a b c d x s           % định nghĩa một số biến đặc trưng
```

```
>> f = [a*x,b*x^2;c*x^3,d*s]   % xây dựng một mảng đặc trưng
```

```
f =
```

```
    [a*x,    b*x^2]
```

```
    [c*x^3,   d*s]
```

```
>> int(f)           % lấy tích phân mảng các phần tử theo đối số x
```

```
ans =
```

```
    [1/2*a*x^2,    1/3*b*x^3]
```

```
    [1/4*c*x^4,    d*s*x]
```

Ví dụ: Giải pháp đặc trưng của một phương pháp tính toán cổ điển

Các bạn Hải, Kiêm và Dũng đang ngồi quan sát trên một mái nhà của một toà cao ốc ở trung tâm Hà Nội trong lúc ăn bữa trưa thì họ chợt phát hiện ra một vật có hình dáng kì lạ trên không ở độ cao 50 m. Họ lấy một quả cà chua chín đỏ ra khỏi chiếc túi đeo sau lưng, tì vào cạnh của mái nhà rồi ném mạnh quả cà chua vào không trung. Quả cà chua được bay lên với vận tốc ban đầu là $v_0 = 20$ m/s. Mái cao 30 m so với mặt đất, thời gian bay của nó là t giây. Hỏi khi nào nó đạt đến độ cao cực đại, độ cao mà quả cà chua đạt tới so với mặt đất? Khi nào thì quả cà chua chạm tới mặt đất? Giả sử rằng không có lực cản của không khí và gia tốc phụ thuộc vào sức hút là không đổi là $a = -9.7536$ m/s².

Chúng ta chọn mặt đất ở độ cao là 0, $y = 0$ là mặt đất và $y = 30$ là đỉnh của toà nhà. Vận tốc tức thời sẽ là $v = dy/dt$, và gia tốc sẽ là $a = d^2y/dt^2$. Do đó nếu lấy tích phân một lần gia tốc, ta sẽ được vận tốc tức thời, còn tích phân vận tốc ta sẽ được độ cao y .

```
>> t = sym('t');           % định nghĩa biến đặc trưng thời gian
```

```
>> digits(5);              % độ chính xác 5 chữ số
```

```
>> a = sym('-9.7536')      % gia tốc đo bằng m/s^2
```

```
a =
```

```
-9.7536
```

```
>> v = nt(a,t)             % vận tốc xem như hàm thời gian
```

```
v =
```

```

-9.7536*t
>> v = v + 20          % ở thời điểm t=0 vận tốc là 20m/s
v =
-9.7536*t + 20
>> y = int(v,t)        % tìm độ cao y ở thời điểm t bằng cách lấy tích phân
y =
-4.8768*t^2+20.*t
>> y = y + 30          % độ cao khi t=0 là 30 m
y =
-4.8768*t^2 + 20.*t + 30

```

Kiểm tra xem kết quả có đúng không, nếu như chúng ta thay $t=0$ vào trong biểu thức, ta được:

```

>> yo = subs(y,t,0)
yo =
30.

```

Kết quả đúng như độ cao quả cà chua trước khi nó được ném

Bây giờ chúng ta đã có vận tốc và vị trí là hàm của thời gian t . Độ cao cực đại khi mà quả cà chua ngừng lên và bắt đầu rơi xuống. Để tìm điểm này, ta tìm giá trị của t khi $v=0$ bằng cách dùng hàm *solve*. Hàm này tìm điểm không của biểu thức đặc trưng, hay nói cách khác, *solve(f)*, trong đó f là hàm của x , tìm x khi cho $f(x) = 0$.

```

>> t_top = solve(v)      % tìm giá trị của t khi v(t)=0
t_top =
2.0505

```

Bởi vì *solve* là một hàm đặc trưng, nó trả lại một hằng đặc trưng (thậm chí nó trông như một số). Bây giờ chúng ta tìm độ cao cực đại, ở thời điểm $t = 2.0505$ s.

```

>> y_max = subs(y, t, t_top) % thay thế t bởi t_top trong y
y_max =
50.505

```

Chú ý rằng hàm *subs* có cùng giá trị như chúng ta làm trước đó khi chúng ta kiểm tra biểu thức y , *subs* sẽ thay biến đặc trưng 2.0505 vào các giá trị t trong biểu thức.

Bây giờ chúng ta tìm thời gian để quả cà chua chạm mặt đất.

```
>> t_splat = solve(y) % quả cà chua chạm mặt đất khi y = 0
```

```
t_splat =
```

```
[-1.1676]
```

```
[ 5.2686]
```

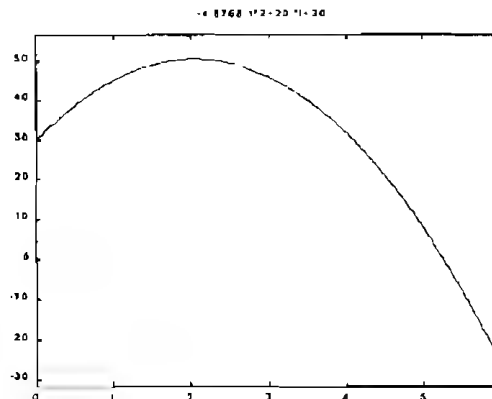
Do kết quả là số âm và quả cà chua không thể chạm đất trước khi nó được ném đi, và nghiệm thứ hai mới là nghiệm có nghĩa. Từ đó suy ra độ cao của quả cà chua ở thời điểm t giây được cho bởi phương trình $y = -9.7536t^2 + 20t + 30$, quả cà chua đạt tới độ cao cực đại 50.505m so với mặt đất và ở thời điểm $t = 2.0505$ s, và nó chạm mặt đất ở thời điểm $t = 5.2686$ s

20.13 Vẽ đồ thị biểu thức đặc trưng

Để có một ý tưởng tốt hơn về chuyện gì xảy ra với quả cà chua, chúng ta vẽ kết quả của trò chơi này (hình 20.1). Gọi vị trí của quả cà chua (độ cao) được miêu tả bằng biểu thức:

$$y = (-4.8768)t^2 + 20t + 30$$

```
>> ezplot(y) % vẽ độ cao quả cà chua
```



Hình 20.1

Như bạn thấy, *ezplot* vẽ đồ thị hàm đặc trưng trong dải $-2\pi \leq t \leq 2\pi$.

20.14 Định dạng và đơn giản hoá biểu thức

Đôi khi MATLAB trả lại một biểu thức đặc trưng rất phức tạp. Một số công cụ có sẵn trợ giúp làm cho biểu thức dễ đọc hơn. Trước tiên đó là hàm *pretty*. Lệnh này hiển thị biểu thức đặc

trung theo một khuôn mẫu tương tự như kiểu toán học. Chúng ta hãy xem sự mở rộng chuỗi Taylor:

```
>> x = sym('x');
>> f = taylor(log(x+1)/(x-5))
f =
-1/5*x+3/50*x^2-41/750*x^3+293/7500*x^4-1207/37500*x^5
>> pretty(f)
```

$$-\frac{1}{5}x + \frac{3}{50}x^2 - \frac{41}{750}x^3 + \frac{293}{7500}x^4 - \frac{1207}{37500}x^5$$

Biểu thức đặc trưng có thể đưa ra dưới nhiều dạng tương tự nhau. MATLAB sử dụng một số lệnh để đơn giản hoá hoặc thay đổi khuôn mẫu trong biểu thức đặc trưng.

```
>> x = sym('x');
>> f = (x^2 - 1)*(x - 2)*(x - 3) % tạo một hàm
f =
(x^2 - 1)*(x - 2)*(x - 3)
>> collect(f) % gộp tất cả các mục như nhau
ans =
x^4 - 5*x^3 + 5*x^2 + 5*x - 6
>> horner(ans)
ans =
-6 + (5 + (5 + (-5*x + x)*x)*x)*x
>> factor(ans) % biểu diễn dưới dạng một đa thức
ans =
(x - 1)*(x - 2)*(x - 3)*(x + 1)
>> expand(f)
ans =
x^4 - 5*x^3 + 5*x^2 + 5*x - 6
```

simplify là một công cụ rất mạnh, mục đích cơ bản là để đơn giản hóa biểu thức dưới nhiều kiểu khác nhau như tích phân và lũy thừa phân số, luật số mũ và hàm log và Bessel, hình học và hàm gamma. Một vài ví dụ sẽ minh họa điều này:

```
>> syms x y a
>> simplify(sin(x)^2 + 3*x + cos(x)^2 - 5)
ans =
-4 + 3*x
>> simplify(log(2*x/y))
ans =
log(2) + log(x/y)
>> simplify((-a^2 + 1)/(1 - a))
ans =
a + 1
```

20.15 Tóm tắt và một số đặc điểm khác

- Biểu thức đặc trưng số phức trong cú pháp MATLAB có thể được trình bày theo một hình mẫu mà ta có thể dễ dàng đọc bằng việc sử dụng hàm *pretty*.
- Có thể có nhiều kiểu tương tự nhau của biểu thức đặc trưng, một số chúng thì dễ dàng sử dụng hơn một số khác trong những tình huống khác nhau. MATLAB đưa ra một số công cụ để thay đổi khuôn dạng trong biểu thức. Đó là:

Hàm	Mô tả
collect	Gom tất cả các mục giống nhau
factor	Biểu diễn dưới dạng một đa thức
expand	Mở rộng tất cả các mục
simplify	Đơn giản hoá các biểu thức
simple	Tìm biểu thức tương đương có chuỗi kí tự ngắn nhất

- Hàm đặc trưng MATLAB có thể được sử dụng để chuyển biểu thức đặc trưng thành phân thức, cho một đa thức hữu tỉ thì *int(f)* sẽ lấy tích phân hàm này, và *diff(f)* sẽ lấy vi phân hàm này. Ví dụ:


```
>> s = sym('s');
>> Y = (10*s^2 + 40*s + 30)/(s^2 + 6*s + 8)
Y =
(10*s^2 + 40*s + 30)/(s^2 + 6*s + 8)
>> diff(int(Y))
ans =
10 - 15/(s + 4) - 5/(s + 2)
>> pretty(ans)
```

$$10 - \frac{15}{s+4} - \frac{5}{s+2}$$

Kỹ thuật này cũng thật là hữu ích khi ta muốn tối giản đa thức trong đó có bậc cao hơn mẫu số.

```
>> x = sym('x');
>> g = (x^3 + 5)/(x^2 - 1)
g =
(x^3 + 5)/(x^2 - 1)
>> diff(int(g))
ans =
x + 3/(-1 + x) - 2/(x + 1)
>> pretty(ans)
```

$$x + \frac{3}{-1+x} - \frac{2}{x+1}$$

20.16 Tự làm

Tìm giá trị của $e^{\pi\sqrt{163}}$ với độ chính xác 18, 29, 30 và 31 số. Chú ý rằng kết quả gần với một giá trị số nguyên nhất, nhưng không hoàn toàn là một số nguyên.

```
>> vpa('exp(pi*sqrt(163))',18)
```

20.17 Giải phương trình

Phương trình đặc trưng có thể được giải bằng công cụ toán học có sẵn trong MATLAB. Một số đồ đã được giới thiệu, một số sẽ được chứng minh ở phần sau.

20.18 Giải phương trình đại số đơn giản

Hàm *solve* gán biểu thức đặc trưng về 0 trước khi giải nó:

```
>> syms a b c x
>> solve(a*x^2 + b*x + c)
ans =
[1/2/a*(-b + (b^2 - 4*a*c)^(1/2))]
[1/2/a*(-b - (b^2 - 4*a*c)^(1/2))]
```

Kết quả là một vectơ đặc trưng mà các phần tử của nó có dạng như trên. Để giải phép toán có chứa dấu bằng, giải một chuỗi có chứa biểu thức:

```
>> solve('a*x^2 + b*x - (-c)')
ans =
[1/2/a*(-b + (b^2 - 4*a*c)^(1/2))]
[1/2/a*(-b -
(b^2 - 4*a*c)^(1/2))]
```

Nếu như bạn muốn giải đối số khác so với biến số mặc định thì bạn có thể khai báo trong *solve* như sau:

```
>> solve(a*x^2 + b*x + c,b)
ans =
-(a*x^2 + c)/x
```

Phép toán có thể giải bằng cách gán biểu thức cho 0. Bây giờ chúng ta sẽ giải $\cos(x)=\sin(x)$ và $\tan(x)=\sin(2x)$ theo x , và quy kết quả của chúng về biến f và t :

```
>> f = solve(cos(x)- sin(x))
f =
1/4*pi
>> t = solve(tan(x)- sin(2*x))
```

t =

```
[ 0]
[ pi]
[ 1/4*pi]
[ -3/4*pi]
```

Kết quả dưới dạng số:

```
>> double(f)
```

ans =

```
0.7854
```

```
>> double(t)
```

ans =

```
0
```

```
3.1416
```

```
0.7854
```

```
-2.3562
```

20.19 Một vài phép toán đại số

Có thể giải vài phép toán cùng một lúc. Câu lệnh `[a1, a2, ..., an] = solve(f1, f2, ..., fn)` giải n phép toán cho các biến mặc định và trả lại kết quả trong $a1, a2, \dots, an$. Tuy nhiên biến mặc định sẽ được lưu trữ. Ví dụ:

```
>> syms x y
```

```
>> [a1 a2] = solve(x^2 + x*y + y - 3, x^2 - 4*x + 3)
```

a1 =

```
[ 1]
```

```
[ 3]
```

a2 =

```
[ 1]
```

```
[-(6*log(3)+lambertw(1/729*log(3)))/log(3)]
```

20.20 Phép toán vi phân

Thông thường phép toán vi phân rất khó giải, MATLAB cung cấp cho bạn một số công cụ mạnh để tìm kết quả của phép toán vi phân.

Hàm *dsolve* sẽ giải các phép toán vi phân và cho ta kết quả. Cú pháp của *dsolve* khác với phần lớn các hàm khác. Đối số của hàm phải là xâu kí tự thay vì biểu thức, ví như xâu chứa một dấu "=". Điều này rõ ràng là khác so với hàm *solve*, mà đối số của nó phải là một biểu thức đặc trưng không có dấu "=".

Phép toán vi phân được nhận ra bằng kí hiệu chữ hoa D và D2, D3, v.v... Bất cứ một chữ nào theo sau Ds đều phụ thuộc vào biến. Phép toán (d^2y/dt^2) được thay bởi chuỗi kí tự 'D2y=0'. các biến độc lập có thể được chỉ ra, hoặc nếu không sẽ mặc định là t

Ví dụ giải phép toán

$(dy/dt) - 1 + 2y^2$:

```
>> clear
```

```
>> dsolve('Dy=1+y^2')
```

```
ans =
```

```
tan(t - C1)
```

trong đó C1 là hằng số. Cũng bài toán trên nhưng cho giá trị ban đầu là $y(0) = 1$ thì sẽ có kết quả sau:

```
>> dsolve('Dy=1+y^2, y(0)=1')
```

```
ans =
```

```
tan(t+1/4*pi)
```

20.21 Một vài phép toán tích phân

Hàm *dsolve* có thể giải nhiều phép toán vi phân cùng một lúc. Khi giải nhiều phép toán vi phân, *dsolve* trả các biến vào một cấu trúc hoặc một vector như *solve* đã làm. Chú ý *dsolve* sắp xếp các biến trước khi độc lập trước khi trả. Ví dụ:

Giải phép toán sau:

$$df/dt = 3f + 4g \quad dg/dt = -4f + 3g$$

```
>> [f,g] = dsolve('Df = 3*f + 4*g, Dg = -4*f + 3*g')
```

```
f =
```

```
exp(3*t)*cos(4*t)*C1 + exp(3*t)*sin(4*t)*C2
```

```
g =
-exp(3*t)*sin(4*t)*C1 + exp(3*t)*cos(4*t)*C2
```

20.22 Ma trận và đại số tuyến tính

Ma trận đặc trưng và vector là các mảng mà phần tử của nó là các biểu thức đặc trưng. chúng có thể được tạo bởi hàm **sym**:

```
>> syms a b c s t
>> A = [a,b,c;b,c,a;c,a,b]
A =
[ a, b, c]
[ b, c, a]
[ c, a, b]
>> G = [cos(t),sin(t);-sin(t),cos(t)]
```

```
G =
[ cos(t), sin(t)]
[ -sin(t), cos(t)]
```

Kích thước của ma trận đặc trưng có thể tìm được bằng hàm chuẩn **size** và **length**. Ví dụ:

```
>> syms a b c d e f
>> S = [a,b,c;d,e,f]
S =
[ a, b, c]
[ d, e, f]
>> h = size(S)
h =
2 3
>> [m,n] = size(S)
```

```
m =
2
```

```
n =
3
```

```
>> length(S)
```

```
ans =
```

```
3
```

Phần tử của mảng đặc trưng cũng được truy nhập tương tự như mảng số

```
>> syms ab cd ef gh
```

```
>> G = [ab,cd,ef,gh]
```

```
G =
```

```
[ ab, cd, ef, gh]
```

```
>> G(1,2)
```

```
ans =
```

```
cd
```

20.23 Phép toán đại số tuyến tính

Phép nghịch đảo và định thức của ma trận được tính bởi hàm: *inv* và *det*

```
>> H = sym(hilb(3))
```

```
H =
```

```
[1, 1/2, 1/3]
```

```
[1/2, 1/3, 1/4]
```

```
[1/3, 1/4, 1/5]
```

```
>> det(H)
```

```
ans =
```

```
1/2160
```

```
>> J = inv(H)
```

```
J =
```

```
[ 9, -36, 30]
```

```
[-36, 192, -180]
```

```
[ 30, -180, 180]
```

```
>> det(J)
```

ans =

2160

20.24 Hàm bước và xung

Hàm *step*, $u(t)$ và hàm *impulse*, $\delta(t)$ thường được dùng trong hệ thống. Hàm bước $Ku(t-a)$ trong đó K là hằng số được định nghĩa như sau: $Ku(t-a) = 0$ nếu $t < a$ và $Ku(t-a) = K$ nếu $T \geq a$.

20.25 Biến đổi Laplace

Phép biến đổi laplace biến đổi từ miền t sang miền s . Hàm của nó như sau:

$$L(s) = \int_0^{\infty} f(t)e^{-st} dt$$

```
>> syms a s t w
```

```
>> f = exp(-a*t)*cos(w*t)
```

```
f =
```

```
exp(-a*t)*cos(w*t)
```

```
>> L = laplace(f,t,s)
```

```
L =
```

```
(s + a)/((s + a)^2 + w^2)
```

```
>> pretty(L)
```

$$\frac{s + a}{2}$$

$s + a) + w$

20.26 Biến đổi Fourier

Hàm biến đổi Fourier và Fourier ngược như sau:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t} dt \quad f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)e^{j\omega t} d\omega$$

MATLAB dùng 'w' thay cho ω trong biểu thức đặc trưng

```
>> syms t w
```

```

>> f=t*exp(-t^2)

f =

    t*exp(-t^2)

>> f=fourier(f,t,w)      % biến đổi fourier sử dụng tham số t và w

f =

    -1/2*i*pi^(1/2)*w*exp(-1/4*w^2)

>> ifourier(f,w,t)      % tìm biến đổi fourier ngược

ans =

    1/2*4^(1/2)*t*exp(-t^2)

>> simplify(ans)

ans =

    t*exp(-t^2)

```


HỘP CÔNG CỤ HỆ THỐNG ĐIỀU KHIỂN

21.1 Sự biểu diễn bằng đồ thị

Phần lớn các công cụ trong **Hộp công cụ hệ thống điều khiển** đều được luận giải dễ hiểu trên cả 2 phương diện hàm truyền và không gian trạng thái. Thêm vào đó hệ thống nhiều đầu vào, nhiều đầu ra (MIMO) được sinh ra từ việc tạo ra ma trận B, C, và D có đối hồi số chiều. Sự biểu diễn hàm truyền MIMO được hình thành do sử dụng ma trận tế bào lưu trữ trong những đa thức hàm truyền tương ứng.

Ví dụ:

```
>> num = { 10, [ 1 10]; -1, [ 3 0 ]};    % mảng tế bào
```

```
>> den = { [ 1 10 ], [ 1 6 10 ]; [ 1 0 ], [ 1 3 3 ]},    %mảng tế bào bậc hai
```

% thay cho hệ thống có 2 đầu vào và 2 đầu ra.

liên tục
$$H(s) = \frac{N(s)}{D(s)} = \frac{N_1 s^m + N_2 s^{m-1} + \dots + N_m s + N_{m+1}}{D_1 s^n + D_2 s^{n-1} + \dots + D_n s + D_{n+1}} \quad m \leq n$$

MATLAB:: num = [N₁ N₂ ... N_{m+1}], den = [D₁ D₂ ... D_{n+1}]

Rời rạc
$$H(z) = \frac{N(z)}{D(z)} = \frac{N_1 z^m + N_2 z^{m-1} + \dots + N_m z + N_{m+1}}{D_1 z^n + D_2 z^{n-1} + \dots + D_n z + D_{n+1}} \quad m \leq n$$

MATLAB: num [N₁ N₂ ... N_{n+1}], den = [D₁ D₂ D_{n+1}]

$$(mẫu z^{-1}) \quad H(z) = \frac{N(z)}{D(z)} = \frac{N_1 + N_2 z^{-1} + \dots + N_m z^{-(m+1)} + N_{m+1} z^{-n}}{D_1 + D_2 z^{-1} + \dots + D_n z^{-(n+1)} + D_{n+1} z^{-n}}$$

MATLAB: num = [N₁ N₂ ... N_{n+1}], den = [D₁ D₂ ... D_{n+1}]

Zero-pole-Gain

Liên tục $H(s) = \frac{N(s)}{D(s)} = K \frac{(s - z_1)(s - z_2) \dots (s - z_m)}{(s - p_1)(s - p_2) \dots (s - p_n)} \quad m < n$

MATLAB: K, Z = [Z₁; Z₂; ... Z_m], P = [P₁; ... P_n]

Rời rạc $H(z) = \frac{N(z)}{D(z)} = K \frac{(z - z_1)(z - z_2) \dots (z - z_m)}{(z - p_1)(z - p_2) \dots (z - p_n)} \quad m \leq n$

MATLAB: K, Z = [Z₁; Z₂; ... Z_m], P = [P₁; ... P_n]

Không gian trạng thái

liên tục $\dot{x} = Ax + Bu$

$$y = Cx + Du$$

MATLAB: A, B, C, D

Rời rạc $x[n+1] = Ax[n] + Bu[n]$

$$y[n] = Cx[n] + Du[n]$$

MATLAB: A, B, C, D

$$\begin{bmatrix} y_1(s) \\ y_2(s) \end{bmatrix} = \begin{bmatrix} \frac{10}{s+10} & \frac{s+10}{s^2+6s+10} \\ \frac{-1}{s} & \frac{3s}{s^2+3s+3} \end{bmatrix} \begin{bmatrix} u_1(s) \\ u_2(s) \end{bmatrix}$$

Có một sự tương quan tự nhiên 1-1 giữa chỉ số mảng tế bào và chỉ số ma trận hàm truyền.

21.2 Đối tượng LTI

MATLAB cung cấp một cách để tóm lược mảng dữ liệu tương quan thành các đối tượng tuyến tính, bất biến theo thời gian, hoặc các đối tượng LTI. Điều này giúp cho việc quản lý chúng được dễ dàng. Ví dụ:

```
>> my_sys= zpk(z, p, k)
```

Zero/ pole / gain from input 1 to output:

11

-

s

Zero / pole / gain from input 2 to output:

$$\frac{3(s+1)}{(s+10)(s+2)}$$

xây dựng một đối tượng LTI zero-pole-gain có tên là my_sys có chứa hệ thống hai đầu vào và một đầu ra. Cũng như vậy.

```
>> H = tf(num, den)
```

Transfer function from input 1 to output...

10

#1:

s+10

-1

#2:

s

Transfer function from input 2 to output ...

s+10

#1:.....

s^2+6 s+10

3s+1

#2:

s^2 + 3 s + 3

tạo một hàm truyền đối tượng LTI từ mảng tế bào num và den nhập vào trước đó. Cũng như vậy nên thống hiện tại hiển thị ở một chế độ dễ hiểu.

Cuối cùng, đối tượng LTI không gian trạng thái được hình thành như sau:

```
>> a = [ 0 1 -2 -4 ]; b = [ 0 1 ]; c = [ 1 1 ]; d = 0;
```

% định nghĩa ma trận không gian trạng thái

```
>> system2=ss(a, b, c, d)
```

a=

	x1	x2
x1	0	1.00000
x	-2.00000	-4.00000

b =

	u1
x1	0
x2	1.00000

c =

	x1	x2
y1	1.00000	1.00000

d=

	u1
y1	0

Hệ thống liên tục theo thời gian

Trong trường hợp này, hệ thống sẽ xác định các thành phần biến gắn với mỗi phần tử và xác nhận hệ thống là liên tục theo thời gian.

Để xây dựng một hệ thống gián đoạn theo thời gian, sử dụng hàm **zpk**, **tf**, và hàm **ss**, bạn nhất thiết phải khai báo chu kì lấy mẫu kèm theo với hệ thống được xem như là một đối số đầu vào cuối cùng Ví dụ:

```
>> dt_sys = tf([ 1 0 2 ], [ 1 -1 ], 0.01)
```

hàm truyền

z+0.

.....

z-1

thời gian lấy mẫu: 0.01

Hệ thống rời rạc theo thời gian này có chu kì lấy mẫu là: 0.01

21.3 Khôi phục dữ liệu

Giả sử đối tượng LTI đã được tạo dựng, thì dữ liệu trong đó có thể tách ra bằng cách sử dụng hàm *tfdata*, *zpkdata*, và *ssdata*. Ví dụ:

```
>> [nz, dz] = tfdata(dt_sys)    % tách ra như là mảng tế bào
nz =
    [1x2 double]
dz =
    [1x2 double]
>> [n z, dz] = tfdata(dt_sys, 'v')    % trích ra như là vector
z =
    [-0.2]
p =
    [ 1]
k =
    1
>> [z, p, k] = zpkdata(dt_sys, 'v')    % trích ra như là vector
z =
    -0.2
p =
    1
k =
    1
>> [a, b, c, d] = ssdata(dt_sys)    % trích ra ma trận không gian trạng
                                     %thái số
a =
    1
b =
    1
```

c =

1.2

d =

1

Nếu như một đối tượng LTI đã được xây dựng thì nó có thể được tách ra theo bất cứ một mẫu nào.

21.4 Sự nghịch đảo đối tượng LTI

Bên cạnh việc tách các đối tượng LTI thành nhiều kiểu khác nhau, chúng còn có thể được chuyển đổi thành các dạng khác nhau bằng cách sử dụng các hàm tự tạo. Ví dụ:

```
>> t = tf(100, [1 6 100]) % xây dựng một hàm truyền.
```

Hàm truyền:

100

.....

$s^2 + 6s + 100$

```
>> sst = ss(t)
```

a =

	x1	x2
x1	-6.00000	-6.25000
x2	16.00000	0

b =

	u1
x1	2.00000
x2	0

c =

	x1	x2
y1	0	3.12500

d =

	u1
y1	0

. Hệ thống liên tục theo thời gian.

```
>> zpkt = zpkt(t)
```

Zero / pole / gain:

100

.....

$(s^2 + 6s + 100)$

21.5 Thuật toán đối tượng LTI

Sử dụng đối tượng LTI cũng cho phép bạn thiết lập thuật toán sơ đồ khối. Ví dụ, hàm truyền lặp của một hệ thống hồi tiếp là $G(s)$, thì hàm truyền lặp gần nhất của là: $T(s) = G(s) / (1 + G(s))$. Trong MATLAB, điều này bắt đầu:

```
>> g = tf(100, [1 6 0]) % hàm truyền lặp
```

Hàm truyền:

100

.....

$s^2 + 6s$

```
>> t = g/(1+g)
```

Hàm truyền:

$100 s^2 + 600 s$

.....

$s^4 + 12 s^3 + 136 s^2 + 600 s$

```
>> t = minreal(t) % thiết lập hàm huỷ pole-zero
```

Hàm truyền:

100

.....

$s^2 + 6s + 100$

21.6 Phân tích hệ thống

Hộp dụng cụ hệ thống điều khiển (**The Control System Toolbox**) có đề cập đến việc phân tích hệ thống số và thiết kế hàm. Để hoàn thiện tài liệu này, hãy xem help trực tuyến. Để hiểu được một số đặc điểm, hãy tham chiếu đến đối tượng LTI open-loop và closed-loop.

```
>> g = zpk([], [0, -5, -10], 100) % hệ thống open-loop
```

Zero/pole/gain:

100

.....

$s(s+5)(s+10)$

>> t = minreal(g/(1+g)) Hệ thống closed-loop

Zero / pole/ gain:

100

.....

$(s+11.38)(s^2 + 3.62s + 8.789)$

Poles của hệ thống này là:

>> pole(t)

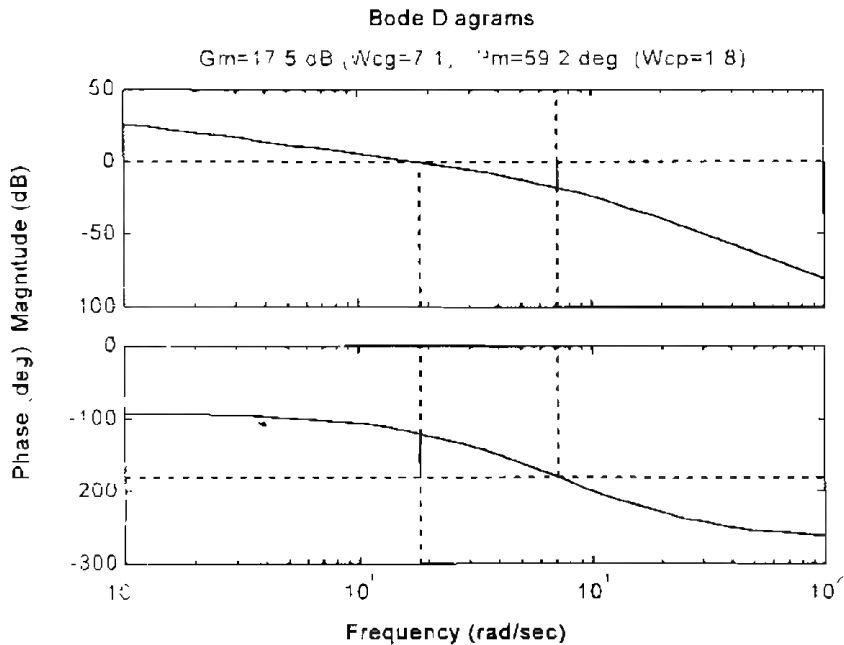
ans =

-11.387

-1.811 + 2.3472i

-1.811 + 2.3472i

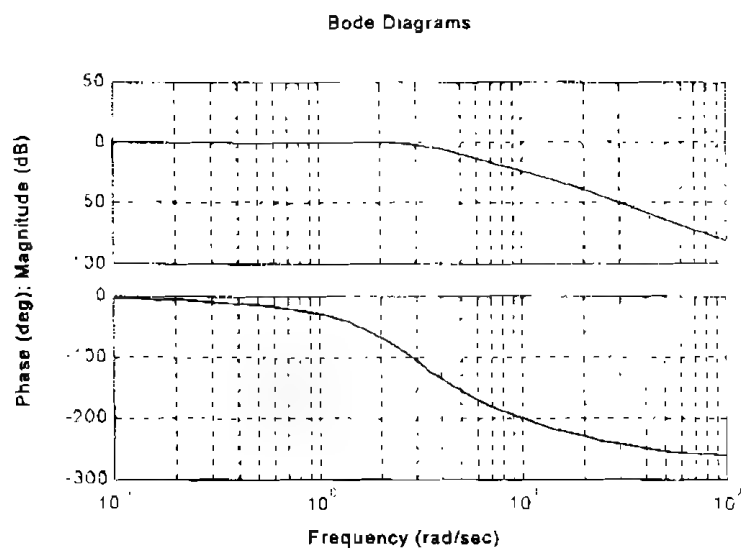
Đồ thị Bode của hệ thống được cho như hình 21.1:



Hình 21.1. Đồ thị Bode của hệ thống

Đồ thị Bode đơn giản của hệ thống closed-loop (hình 21.2):

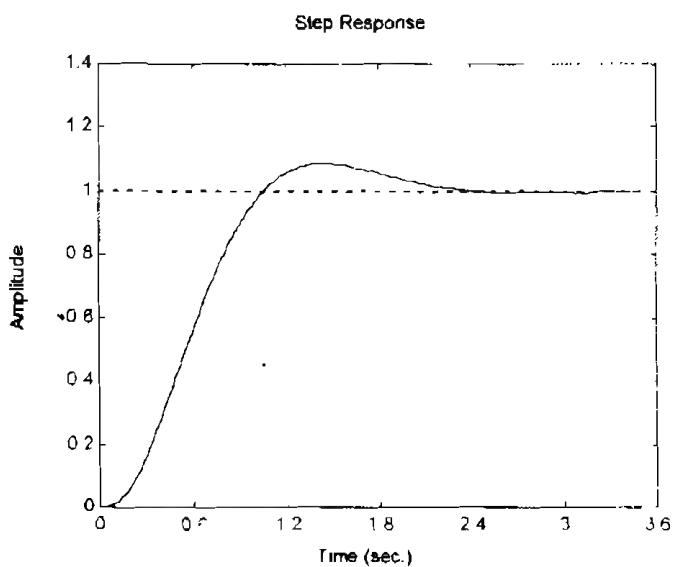
```
>> bode(t)
```



Hình 21.2

Đáp ứng xung của hệ thống như trên hình 21.3.

```
>> step(t)
```



Hình 21.3 Đáp ứng xung của hệ thống

Ngoài các phương pháp nêu trên, hộp công cụ hệ thống điều khiển còn đưa ra thêm cho bạn lệnh trợ giúp **ltview**. Hàm này cho phép bạn lựa chọn các đối tượng LTI từ cửa sổ lệnh và quan sát các đáp ứng khác nhau trên màn hình.

21.7 Danh sách các hàm của hộp công cụ hệ thống điều khiển

Sự hình thành các kiểu LTI	
ss	Xây dựng kiểu không gian trạng thái
zpk	Xây dựng kiểu zero-pole-gain
tf	Xây dựng kiểu hàm truyền
dss	Chỉ rõ kiểu hóa pháp không gian trạng thái
fit	Chỉ rõ bộ lọc số
set	Thiết lập hoặc sửa đổi đặc tính của LTI
lt.props	Trợ giúp chi tiết cho đặc tính TTI

Phân tách dữ liệu	
ssdata	Tách ma trận không gian trạng thái
zpkdata	Tách dữ liệu zero-pole-gain
tfdata	Tách tử số và mẫu số
dssdata	Chỉ ra version của data
get	Truy nhập đặc tính giá trị của LTI

Đặc tính của các loại	
class	Kiểu model ('ss', 'zpk', or 'tf')
size	Số chiều của đầu vào/ đầu ra
isempty	True cho kiểu LTI rỗng
isct	True cho kiểu liên tục theo thời gian
isdt	True cho loại gián đoạn theo thời gian

isproper	True cho kiểu LTI cải tiến
issiso	True cho hệ thống một đầu vào/ một đầu ra
isa	Kiểm tra Loại LTI được đưa ra

Sự nghịch đảo	
ss	Chuyển đổi thành không gian trạng thái
zpk	Chuyển đổi thành zero-pole-gain
tf	Chuyển đổi thành hàm truyền
c2d	Chuyển đổi từ liên tục sang gián đoạn
d2d	Lấy mẫu lại hệ thống rời rạc hoặc thêm độ trễ đầu vào

Các phép toán	
+ và -	Cộng và trừ hệ thống LTI (mắc song song)
*	Nhân hệ thống LTI (mắc nối tiếp)
\	Chia trái $\text{sys1} \backslash \text{sys2}$ nghĩa là: $\text{inv}(\text{sys1}) * \text{sys2}$
/	Chia phải $\text{sys1} / \text{sys2}$ có nghĩa $\text{sys1} * \text{inv}(\text{sys2})$
.	Hoán vị ngược
:	Hoán vị đầu vào/đầu ra
[...]	Sự kết nối hệ thống LTI ngang/ dọc
inv	Nghịch đảo hệ thống LTI

Động học	
pole, eig	Hệ thống poles
tzero	Sự truyền hệ thống các số 0
pzma	Biểu đồ Pole-Zero
dcgain	Định hướng DC (tần số thấp)

norm	Chỉ tiêu hệ thống LTI
covar	Covar of response lên nhiễu trắng
damp	Tần số tự nhiên và sự suy giảm cực hệ thống
esort	Sắp xếp cực tính liên tục bởi phần thực
dsort	Sắp xếp cực tính rời rạc bởi biên độ
pade	Xấp xỉ pade của thời gian trễ

Đáp ứng thời gian	
step	Đáp ứng bước
impulse	Đáp ứng xung
initial	Đáp ứng hệ thống không gian trạng thái với trạng thái khởi tạo
lsim	Đáp ứng đầu vào tùy ý
Ltview	Đáp ứng phân tích GUI
gensig	Phát sinh tín hiệu đầu vào cho lsim
stepfun	Phát sinh đầu vào đơn vị -bước

Đáp ứng tần số	
bode	Đồ thị Bode của đáp ứng tần số
sigma	Đồ thị giá trị tần số duy nhất
nyquist	Đồ thị Nyquist
nichols	Biểu đồ Nichols
ltview	Đáp ứng phân tích GUI
evalfr	Đáp ứng tần số tại một tần số nhất định
margin	Giới hạn pha và tăng ích

Liên kết hệ thống	
append	Nhóm hệ thống LTI bởi việc thêm các đầu ra và đầu vào
parallel	Kết nối song song (tương tự overload +)
series	Kết nối nối tiếp (tương tự overload *)
feedback	Kết nối hồi tiếp hai hệ thống
star	Tích số star(kiểu liên kết LFT)
connect	Chuyển hoá từ kiểu không gian trạng thái sang đặc tính biểu đồ khối

Dụng cụ thiết kế cổ điển	
rlocus	Quỹ tích nghiệm
acker	Sự thay thế cực SISO
place	Sự thay thế các MIMO
estime	Khuôn dạng bộ đánh giá

Công cụ thiết kế LQG	
lqr, dlqr	Bộ điều chỉnh hồi tiếp và phương trình bậc hai tuyến tính
lqry	Bộ điều chỉnh LQ với đầu ra phụ
lqrd	Bộ biến đổi LQ rơi rạc sang liên tục
kalman	Bộ đánh giá Kalman
lqgreg	Bộ biến đổi LQG được đưa ra từ độ tăng ích LQ và bộ đánh giá Kalman

Giải quyết phép toán ma trận	
lyap	Giải phương trình Lyapunov liên tục
dlyap	Giải phương trình Lyapunov rời rạc

care	Giải phương trình đại số Riccati liên tục
dare	Giải phương trình đại số Riccati rời rạc

Sự biểu diễn	
crtldemo	Giới thiệu đến hộp công cụ hệ thống điều khiển
jetdemo	Thiết kế kinh điển bù chống suy giảm âm của phương tiện vận chuyển trực thăng
diskdemo	Thiết kế bộ điều khiển số ổ đĩa cứng
milldemo	Điều khiển LQG SISO và MIMO của hệ thống cân thép tròn
kalmdemo	Thiết kế bộ lọc Kalman và mô phỏng

HỘP DỤNG CỤ XỬ LÝ TÍN HIỆU

22.1 Chức năng của hộp công cụ xử lý tín hiệu

Hộp công cụ Xử lý tín hiệu gồm các công cụ xây dựng trên môi trường tính toán số của MATLAB. Hộp công cụ cung cấp một lượng lớn các toán tử xử lý tín hiệu từ việc tạo ra các dạng tín hiệu đến việc thiết kế và thực hiện các bộ lọc, mô tả các tham số cho đến việc phân tích phổ. Hộp công cụ chia làm hai phần, các hàm dòng lệnh với các nhóm lệnh thực hiện phân tích và lọc số và tương tự, phân tích phổ và mô hình hoá các dạng sóng dự đoán tuyến tính. Phần thứ hai là giao diện tương tác người sử dụng dạng đồ hoạ.

Các tính năng chủ yếu của hộp thoại là: Các hàm công cụ, chủ yếu là các thuật toán viết thành các M- file. Lọc và biến đổi FFT là hai chức năng quan trọng nhất của xử lý số tín hiệu đã được xây dựng bên trong MATLAB mà không nằm ở hộp công cụ. Hộp công cụ có sử dụng đến nhiều chức năng của ngôn ngữ như tính toán ma trận, số phức, tìm nghiệm của đa thức, đồ hoạ, các tín hiệu và hệ thống. Các hàm chức năng của hộp công cụ tập trung vào là rõ sự khác biệt về tính chất số và tương tự của các bộ lọc và tín hiệu. Các kết quả thể hiện được theo nhiều cách như hàm truyền, trạng thái không gian, điểm cực – không... và cho phép chuyển đổi giữa chúng.

22.2 Biểu diễn tín hiệu

Cấu trúc dữ liệu trung tâm của MATLAB là mảng số, một tập hợp có thứ tự các số thực hoặc số phức biểu diễn trong hai hay nhiều chiều. Các đối tượng dữ liệu chính của xử lý tín hiệu (các tín hiệu một chiều hoặc các chuỗi, các tín hiệu đa kênh và các tín hiệu hai chiều) có bản chất tự nhiên phù hợp với sự biểu diễn của mảng. Một cách biểu diễn chuỗi là liệt kê các phần tử của nó, ví dụ lệnh:

```
>> x = [4 3 7 -9 1]
```

```
x =
```

```
4    3    7   -9    1
```

sẽ tạo một chuỗi 5 phần tử như một vector hàng. Chuyển vị của x, sẽ tạo ra một vector cột:

```
>> x' = x'
```

```
x =
    4
    3
    7
   -9
    1
```

Biểu diễn dạng cột thích hợp cho các tín hiệu đơn kênh bởi vì sự mở rộng nó một cách tự nhiên sẽ phù hợp cho trường hợp đa kênh. Với tín hiệu đa kênh, mỗi cột của ma trận sẽ biểu diễn một kênh, mỗi dòng của ma trận sẽ biểu diễn cho một mẫu của tín hiệu. Ví dụ một tín hiệu y gồm 3 kênh x , $2x$, và x/π sẽ có kết quả biểu diễn là:

```
>> y = [x 2*x x/pi]
y =
    4.0000    8.0000    1.2732
    3.0000    6.0000    0.9549
    7.0000   14.0000    2.2282
   -9.0000  -18.0000   -2.8648
    1.0000    2.0000    0.3183
```

22.3 Các tín hiệu đa kênh

Sử dụng các cú pháp mảng tiêu chuẩn của MATLAB để làm việc với các tín hiệu đa kênh. Ví dụ để tạo một tín hiệu z gồm 3 kênh tín hiệu 1 , $t.^2$, $\text{square}(4*t)$ ta dùng lệnh:

```
>> z = [t t.^2 square(4*t)];
```

Bạn có thể tạo một mẫu tín hiệu đơn v cho một kênh của tín hiệu đa kênh bởi toán tử nhân ngoài. Ví dụ một vector cột 6 thành phần trong đó thành phần đầu tiên bằng 1 còn các thành phần khác bằng không được tạo ra như sau:

```
>> a = [1 zeros(1,5)]';
```

```
a =
```

```
    1
    0
    0
    0
    0
    0
```


Để ghép tạo ra các kênh có cùng một tín hiệu từ kênh ban đầu, ta thực hiện toán tử (:). Ví dụ từ tín hiệu kênh a ta tạo tín hiệu đa kênh c gồm 3 kênh a:

```
>> c = a(:,ones(1,3));
```

```
c =
```

```
1  1  1
0  0  0
0  0  0
0  0  0
0  0  0
0  0  0
```

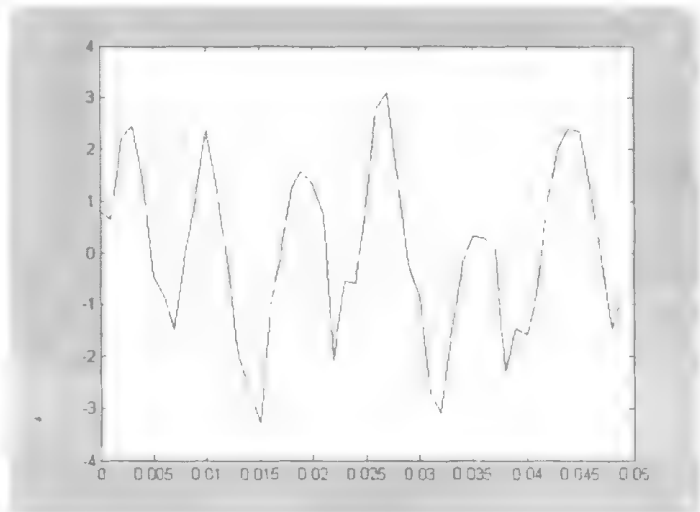
22.4 Tạo dạng tín hiệu

Để tạo dạng tín hiệu theo thời gian, trước hết cần tạo một vector thể hiện thời gian cơ sở. Ví dụ để phát ra một tín hiệu với tần số lấy mẫu 1000 Hz, vector thời gian có dạng

```
>> t = (0:0.001:1)';
```

Ở đây toán tử (') của MATLAB tạo ra một vector hàng có 1001 phần tử thể hiện thời gian chạy theo từng bước 1ms từ 0 đến 1s. Toán tử (') chuyển vector hàng thành vector cột. Dấu (:) để MATLAB thực hiện lệnh mà không hiển thị kết quả. Với mảng t đã có, ta có thể tạo một tín hiệu là tổng hai hình sin có tần số 50 Hz và 120 Hz:

```
>> y =
sin(2*pi*50*t) +
2*sin(2*pi*120*t);
```



Hình 22.1

Biến số mới y tạo ra từ vector t cũng là một vector 1001 phần tử. Bạn cũng có thể tạo tín hiệu nhiễu từ nhiễu trắng bởi hàm randn('state',0) và vẽ đồ thị của tín hiệu trong 50 mẫu đầu tiên:

```
>> yn = y + 0.5*randn(size(t));
>> plot(t(1:50), yn(1:50))
```

22.5 Các tín hiệu cơ bản

Với khả năng của một ngôn ngữ lập trình, MATLAB có thể tạo ra nhiều loại tín hiệu khác nhau. Hàm xung đơn vị, hàm bước nhảy và hàm tuyến tính là các tín hiệu cơ sở của xử lý tín hiệu, có thể được tạo ra một cách đơn giản trong MATLAB:

Tạo xung đơn vị:

```
>> t = (0:0.001:1)';
>> y = [1; zeros(99,1)]; % impulse
```

Tạo xung bước nhảy:

```
>> t = (0:0.001:1)';
>> y = ones(100,1);
```

Tạo xung tuyến tính:

```
>> t = (0:0.001:1)';
>> y = t;
```

Các dạng xung khác:

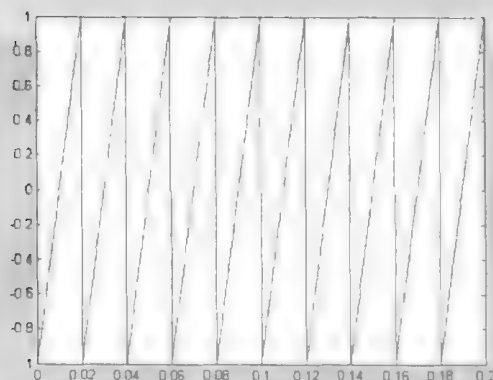
```
>> t = (0:0.001:1)';
>> y = t.^2;
>> y = square(4*t);
```

22.6 Các tín hiệu tuần hoàn thông dụng

Hàm sawtooth sẽ tạo ra một chuỗi xung tam giác với biên độ ± 1 . Các đỉnh cực đại của tín hiệu xác định tại các giá trị bội số của 2π . Để tạo tín hiệu tuần hoàn trong khoảng 1s, ta chọn sử dụng hàm với đối số $(2\pi \cdot t)$. Ví dụ tạo tín hiệu xung răng cưa tần số 50 Hz trong khoảng thời gian 1.5s với tần số lấy mẫu $f_s = 10000$:

```
>> fs = 10000;
>> t = 0:1/fs:1.5;
>> x = sawtooth(2*pi*50*t);
plot(t,x); axis([0 0 2 -1 1]);
```

Kết quả:

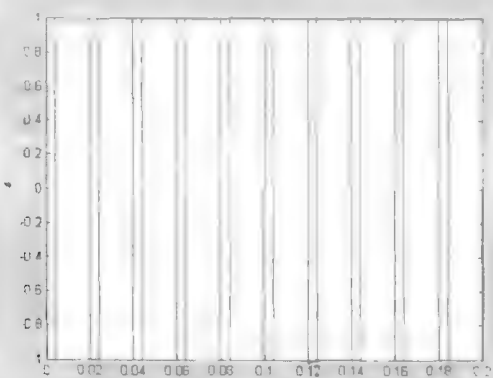


Hình 22.2

Hàm `square(t)` tạo ra tín hiệu xung vuông giống như hàm `sin(t)` nhưng có các giá trị biên độ ± 1 thay cho dạng tín hiệu hình sin. Hàm `square(t,duty)` tạo ra tín hiệu với độ rộng xung xác định với hệ số `duty`. Hệ số `duty` xác định mức phần trăm của chu kỳ trong đó tín hiệu có giá trị +1. Ví dụ tạo tín hiệu xung vuông tần số 50 Hz trong khoảng thời gian 1.5s với tần số lấy mẫu $f_s = 10000$, độ rộng là 20%:

```
>> fs = 10000;
>> t = 0:1/fs:1.5;
>> x = square(2*pi*50*t,20);
plot(t,x); axis([0 0.2 -1 1]);
```

Kết quả:



Hình 22.3

22.7 Các tín hiệu không chu kỳ

Hộp công cụ cũng cung cấp một số hàm tạo tín hiệu không có chu kỳ. hàm `gauspuls` tạo một tín hiệu xung hình sin điều chế Gaussian với tần số trung tâm và bằng thông xác định:

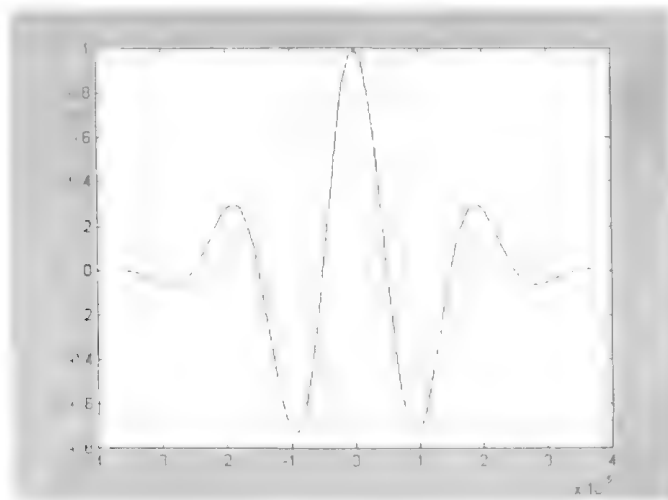
```
>> tc = gauspuls('cutoff',50e3,0.6,[],-40);
```

```
t = -tc : 1e-6 : tc;
```

```
yi = gauspuls(t,50e3,0.6);
```

```
plot(t,yi)
```

Kết quả:



Hình 22.4

Hàm `chirp` tạo ra một tín hiệu cosin có tần số biến đổi tuyến tính, các tham số tùy chọn để xác định phương pháp quét góc pha ban đầu. Ví dụ sau cho tín hiệu tuyến tính chirp trong thời gian 2s với tần số lấy mẫu 1 kHz, khởi đầu từ tần số 0 và đạt tần số 150 Hz tại thời điểm 1 s:

```
>> t = 0:1/1000:2;
```

```
y = chirp(t,0,1,150);
```

Để vẽ đồ thị phổ - thời gian của tín hiệu dùng lệnh `specgram`

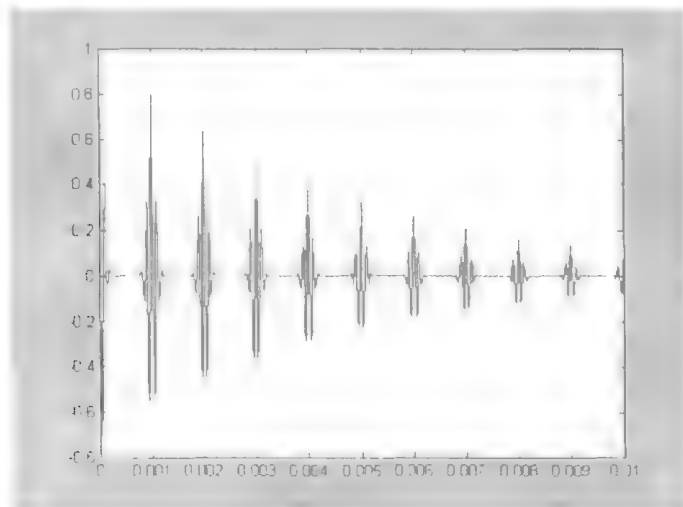
22.8 Hàm chuỗi xung

Hàm `pulstran` tạo ra một chuỗi xung từ các xung liên tục hay các mẫu xung đơn lẻ. Ví dụ sau đây là một chuỗi xung được tạo bởi tổng các mẫu trễ của xung Gaussian. Tần

số lấy mẫu là 50 kHz, độ dài mỗi xung là 10 ms, và tốc độ lặp 1 kHz, Giá trị D xác định mức trễ giữa mỗi khoảng lặp và mức suy giảm:

```
>> T = 0:1/50E3:10E-3;
>> D = [0:1/1E3:10E-3;0.8.^(0:10)]';
>> Y = puistran(T,D,'gauspuls',10E3,0.5);
>> plot(T,Y)
```

Kết quả:



Hình 22.5

22.9 Hàm Sinc

Hàm sinc(x) thực hiện tính biểu thức toán của hàm sinc với đầu vào là vector hoặc ma trận x. Biểu thức của hàm được cho là:

$$\text{sinc}(t) = \begin{cases} 1, & t = 0 \\ \frac{\sin(\pi t)}{\pi t}, & t \neq 0 \end{cases}$$

Hàm sinc là biến đổi Fourier ngược của xung vuông có độ rộng và biên độ bằng 1

$$\text{sinc}(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\omega t} d\omega$$

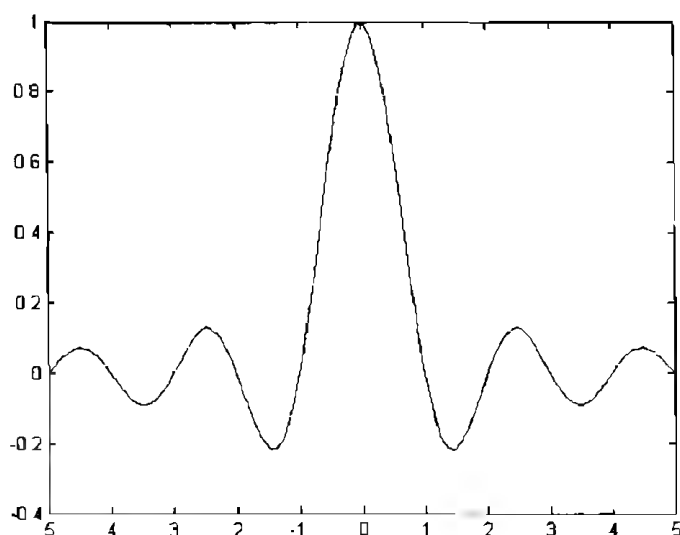
Các lệnh sau sẽ vẽ hàm sinc trong khoảng -5 to 5:

```
>> x = linspace(-5,5);
```

```
>> y = sinc(x);
```

```
>> plot(x,y)
```

Kết quả:



Hình 22.6

Các hàm tín hiệu khác:

dirc

Tính hàm sinc

tripuls

Tạo xung tam giác có chu kỳ

22.10 Nhập xuất dữ liệu với hộp công cụ

Các ví dụ trên cho thấy có thể nạp dữ liệu vào MATLAB bằng một trong hai cách: nhập dữ liệu trực tiếp bằng bàn phím, hoặc dùng các hàm của MATLAB như `sin`, `cos`, `sawtooth`, `square`, hoặc `sinc`. Một số ứng dụng có thể phải sử dụng đến các dữ liệu ở bên ngoài môi trường MATLAB. Tùy thuộc vào khuôn dạng dữ liệu sử dụng, ta có thể nhập dữ liệu từ file ASCII hoặc file MAT với lệnh `load`. Có thể đọc dữ liệu vào MATLAB với các hàm I/O cấp thấp khác như `fopen`, `fread`, và `fscanf`. Có thể phát triển một MEX-file để đọc dữ liệu. Các phương pháp khác cũng rất hữu ích chẳng hạn như dùng các ngôn ngữ bậc cao (ví dụ bằng C hay Fortran) để chuyển đổi các dữ liệu của bạn vào dạng file MAT-xem thêm các hàm MATLAB mở rộng giao tiếp API. Các kỹ thuật tương tự cũng được sử dụng để xuất dữ liệu từ môi trường MATLAB.

22.11 Thực hiện tích chập

Tích chập của hai tín hiệu $x(n)$ và $h(n)$ được định nghĩa:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{k=\infty} x(k)h(n-k)$$

Nếu $x(n)$ và $y(n)$ có chiều dài hữu hạn lần lượt là M và N thì $y(n)$ cũng có chiều dài hữu hạn và bằng $M+N-1$. Hàm conv thực hiện tính tích chập một chiều của hai tín hiệu. Ví dụ:

```
>> conv([1 1 1],[1 1 1])
```

```
ans =
```

```
1 2 3 2 1
```

Tích chập hai chiều được thực hiện bởi hàm conv2.

22.12 Thực hiện lọc trong miền thời gian

Trong miền thời gian, mỗi bộ lọc số được đặc trưng bởi một hàm đáp ứng xung $h(k)$, tín hiệu đầu của bộ lọc là tích chập của tín hiệu đầu vào với hàm đáp ứng xung. Ví dụ với $h = [1 \ 1 \ 1]/4$, tín hiệu vào $x = \text{randn}(5,1)$; % vector ngẫu nhiên với chiều dài 5, tín hiệu đầu ra là:

```
>> h = [1 1 1]/4
```

```
x = randn(1,5)
```

```
y = conv(h,x)
```

Kết quả:

```
h =
```

```
0.2500 0.2500 0.2500 0.2500
```

```
x =
```

```
1.1244 0.9794 -1.3164 -0.0232 0.1345
```

```
y =
```

```
0.2811 0.5259 0.1968 0.1910 -0.0564 -0.3013 0.0278 0.0336
```

22.13 Thực hiện bộ lọc trong miền z

Biến đổi z của tín hiệu ra và tín hiệu vào có quan hệ:

$$Y(z) = H(z)X(z) = \frac{b(1) + b(2)z^{-1} + \dots + b(n+1)z^{-n}}{a(1) + a(2)z^{-1} + \dots + h(m+1)z^{-m}} X(z)$$

Trong đó $H(z)$ được gọi là hàm truyền đạt của hệ thống. Các giá trị $b(i)$ và $a(i)$ là các hệ số của bộ lọc và bậc của bộ lọc xác định bởi giá trị cực đại của m và n . MATLAB chứa

các hệ số trong hai vector hàng, một cho tử số và một cho mẫu số. Các bộ lọc được gọi tên theo các hệ số và số bậc của tử và mẫu:

Nếu $n = 0$ (b là một hệ số tỷ lệ), bộ lọc gọi là đáp ứng xung vô hạn (IIR), chỉ có điểm cực, hay bộ lọc tự hồi quy (autoregressive -AR).

Nếu $m = 0$ (a là một hệ số tỷ lệ), bộ lọc gọi là đáp ứng xung hữu hạn (FIR), chỉ có điểm không, hay bộ lọc trung bình chuyển dịch (moving average -MA).

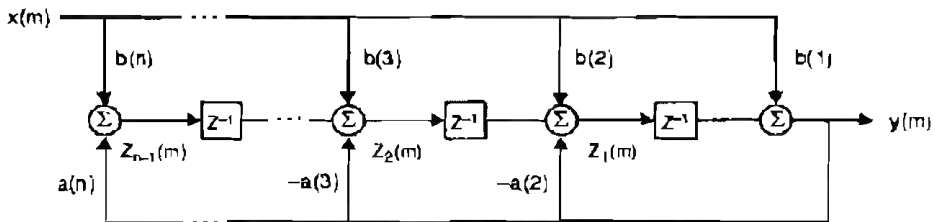
Nếu m và n đều khác không, bộ lọc gọi là IIR, bộ lọc cực – không, hay bộ lọc ARMA (autoregressive moving average).

Các từ phân loại AR, MA và ARMA thường gắn liền với các bộ lọc và xác định quá trình lọc của chúng.

22.14 Thực hiện lọc bằng hàm filter

Hàm $y = \text{filter}(b,a,X)$ lọc các dữ liệu trong vector X bởi bộ lọc được mô tả bởi các hệ số của tử số nằm trong vector b và các hệ số của mẫu số nằm trong vector a . Nếu $a(1)$ khác 1, bộ lọc sẽ chuẩn hoá các hệ số khác bởi phép chia cho $a(1)$. Nếu $a(1)$ bằng 0, bộ lọc trả lại một lỗi.

Hàm $y = \text{filter}(b,a,X)$ lọc các dữ liệu trong vector X với bộ lọc được mô tả bởi các hệ số trong các vector a và b . Nếu $a(1)$ khác 1, bộ lọc sẽ chia tất cả các hệ số cho $a(1)$. Nếu $a(1)$ bằng không, bộ lọc sẽ báo lỗi. Hàm filter cho các đầu ra trực tiếp từ đầu vào theo phương trình sai phân, biểu diễn bởi cấu trúc sau:



Hình 22.7

hay: $y(n) = b(1)*x(n) + b(2)*x(n-1) + \dots + b(nb+1)*x(n-nb)$

$a(2)*y(n-1) - \dots - a(na+1)*y(n-na)$.

Ví dụ: bộ lọc trung bình bề rộng cửa sổ 5 phần tử:

```
>>data = [1:0.2:4]';
```

```
windowSize = 5;
```

```
y = filter(ones(1>windowSize)/windowSize,1,data);
```

```
>> y'
```

Kết quả:

ans =

Columns 1 through 8

0.2000 0.4400 0.7200 1.0400 1.4000 1.6000 1.8000 2.0000

Columns 9 through 16

2.2000 2.4000 2.6000 2.8000 3.0000 3.2000 3.4000 3.6000

Đây là bộ lọc FIR bậc 5 với các hệ số $b(i)$ đều bằng $1/5$.

Vì hàm filter tính toán đầu ra theo từng nhịp nên có các khoảng trễ giữa đầu vào và đầu ra. Hàm filter có thể nhận thêm tham số thứ 4 là các giá trị khởi tạo đầu ra, và các giá trị đầu ra cuối cùng cũng nhận lại được: $[y,zf] = \text{filter}(b,a,x,zi)$, ở đây, y chứa các mẫu lọc đúng với thứ tự các mẫu đầu vào, zf nhận được các giá trị đầu ra trễ khi mẫu đầu vào cuối cùng ra khỏi bộ lọc, zi là các mẫu khởi tạo đầu ra. Ví dụ:

```
>> [y,zf] = filter(ones(1>windowSize)/windowSize,1,data);
```

```
>> zf
```

Kết quả:

zf =

2.9600

2.2800

1.5600

0.8000

Hàm filter để lọc tín hiệu rời rạc. Sử dụng các chức năng khác của hộp công cụ để phân tích các đặc tính của bộ lọc như: đáp ứng xung, đáp ứng pha và tần, trễ nhóm và phân bố của các điểm không và điểm cực.

22.15 Đáp ứng xung

Đáp ứng xung của một bộ lọc số là tín hiệu đầu ra khi đầu vào là xung đơn vị. Ví dụ đáp ứng xung của một bộ lọc đơn giản có $b = 1$ và $a = [1 - 0.9]$:

```
>> imp = [1; zeros(49,1)];
```

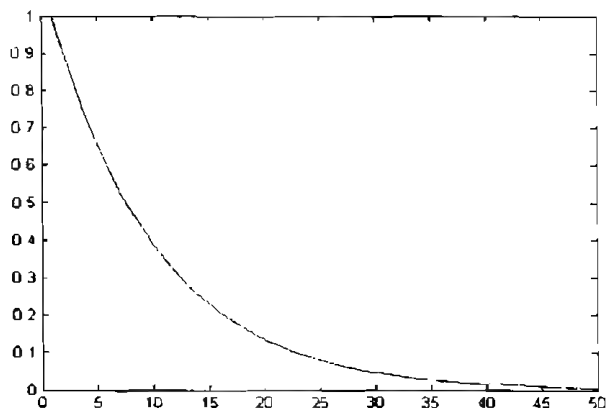
```
>> b = 1;
```

```
>> a = [1 - 0.9];
```

```
>> h = filter(b,a,imp)
```

```
>> figure, plot(h);
```

Kết quả:



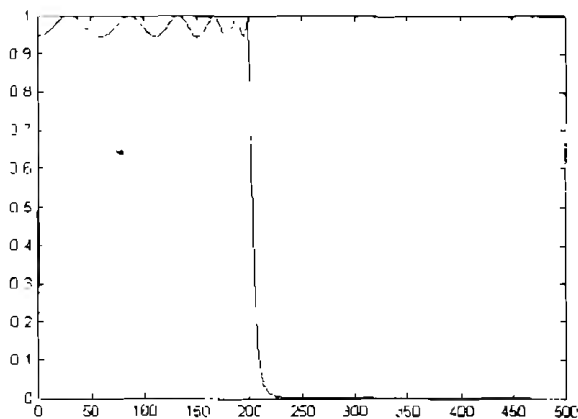
Hình 22.8

22.16 Đáp ứng tần số

Hộp công cụ cho phép bạn biểu diễn các kết quả phân tích trong miền tần số của các bộ lọc tương tự và số. Lệnh `[h,w] = freqz(b,a,p)` sẽ tính đáp ứng tần số phức tại p điểm từ các vector hệ số a và b . Hàm `freqz` trả lại đáp ứng tần số phức trong vector h và các điểm tần số thực nằm trong vector w theo đơn vị rad/s. Hàm `freqz` có thể nhận các tham số khác như tần số lấy mẫu hay một vector chứa các điểm tần. Ví dụ sau thể hiện đáp ứng tần số của bộ lọc Chebyshev loại I bậc 12, hàm `freqz` xác định với tần số lấy mẫu $f_s = 1000$:

```
>> [b,a] = cheby1(12,0.5,200/500);
>> [h,f] = freqz(b,a,256,1000);
>> plot(f,abs(h));
```

Kết quả



Hình 22.9

Ta thấy hàm trả lại giá trị f là tần số Nyquist bằng một nửa tần số lấy mẫu ($f_s/2$).

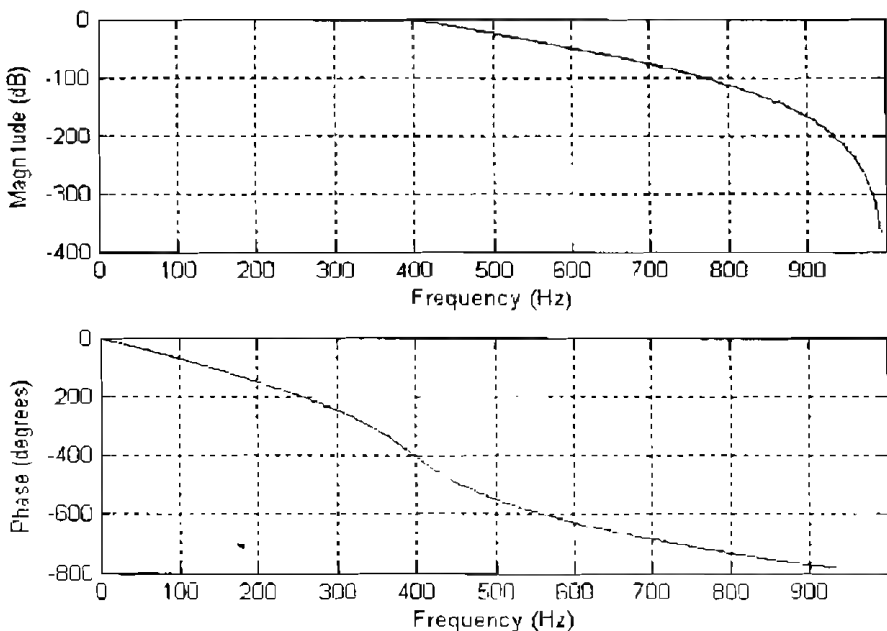
Khi danh sách tham số bao gồm tần số lấy mẫu thì hàm `freqz` trả lại vector f gồm 256 điểm tần số giữa 0 và tần số $f_s/2$, và các đáp ứng biên độ, pha được tính theo các điểm tần này. Chú ý. Hộp công cụ thường sử dụng tần số Nyquist (một nửa tần số lấy mẫu) làm tần số chuẩn hoá. Các tham số về tần số cắt của các bộ lọc cơ bản thường được thiết kế theo tỷ lệ với tần số chuẩn hoá này. Ví dụ, với hệ thống số có tần số lấy mẫu là 1000Hz, tần số Nyquist sẽ là $1000/2 = 500\text{Hz}$, nếu tần số cắt là 300Hz thì giá trị chuẩn là $300/500 = 0.6$. Để chuyển đổi tần số chuẩn hoá thành tần số góc ta nhân tần số chuẩn với π , để chuyển thành tần số thực thì chỉ cần nhân với tần số Nyquist.

Nếu bạn gọi hàm `freqz` không có các tham số đầu ra thì nó sẽ vẽ các đặc tuyến biên độ và pha theo tần số. Ví dụ các lệnh sau sẽ vẽ đáp ứng tần số của bộ lọc thông thấp Butterworth bậc 9 với tần số cắt là 400 Hz, với tần số lấy mẫu là 2000Hz:

```
>> [b,a] = butter(9,400/1000);
```

```
freqz(b,a,256,2000);
```

Kết quả:



Hình 22.10

Hàm `freqz` cũng có thể nhận một vector các điểm tần số chuẩn hoá và tính đáp ứng tần số trên các điểm đó. Ví dụ:

```
>> w = linspace(0,pi);
```

```
>> h = freqz(b,a,w);
```

sẽ tính đáp ứng tần số phức tại các điểm tần w cho bộ lọc xác định bởi các vector b và a . Các điểm tần có thể biến đổi từ 0 đến 2π .

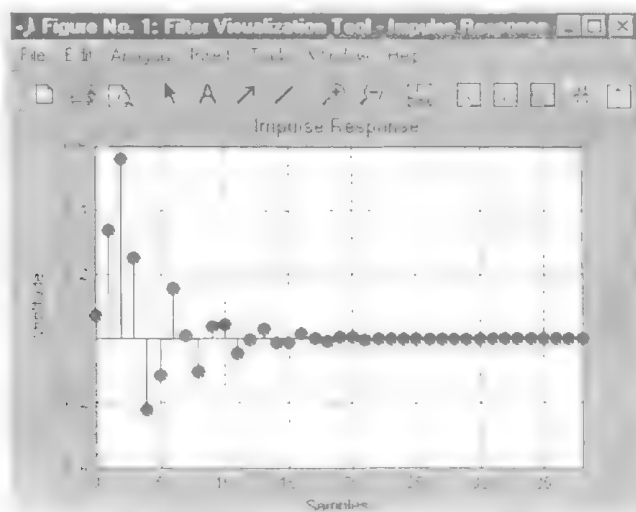
22.17 Hiển thị đáp ứng của hệ thống bằng lệnh fvtool

MATLAB cung cấp các hàm cho phép tách các đáp ứng biên độ và pha từ vector đáp ứng tần số h . Hàm `abs` trả lại đáp ứng biên độ và hàm `angle` trả lại đáp ứng pha tính theo đơn vị rad. Ví dụ, hiển thị các đáp ứng tần số của bộ lọc Butterworth thông thấp bậc 6:

```
>> [b,a] = butter(6,300/500);
```

```
>> fvtool(b,a);
```

Hàm `fvtool` hiển thị một figure cho phép người sử dụng chọn các loại đáp ứng tần số khác nhau như biên độ, pha, trễ nhóm, đáp ứng xung, đáp ứng hàm bước nhảy, phân bố cực-không và các hệ số a, b .



Hình 22.11

22.18 Trễ nhóm và trễ pha

Trễ nhóm của một bộ lọc là phép đo độ trễ trung bình của bộ lọc như một hàm của tần số. Trễ nhóm được xác định là giá trị âm của đạo hàm bậc nhất đáp ứng pha của bộ lọc:

$$\tau_g(\omega) = -\frac{d\theta(\omega)}{d\omega}$$

Trong đó θ là góc pha của đáp ứng tần số phức bộ lọc. MATLAB tính trễ nhóm bởi hàm `[gd,w] = grpdelay(b,a,n)`. Trong đó `gd` là vector trễ, `w` là vector tần số, `n` là số điểm tính.

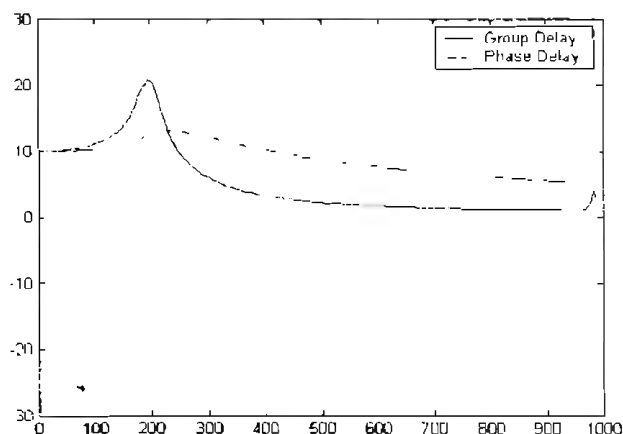
Trễ pha của một bộ lọc là giá trị âm của góc pha chia cho tần số:

$$\tau_p(\omega) = -\frac{d\theta(\omega)}{d\omega}$$

Ví dụ sau sẽ vẽ các giá trị trễ pha và trễ nhóm của bộ lọc thông thấp butterworth bậc 10 trên cùng một đồ thị:

```
>> [b,a] = butter(10,200/1000);
>> gd = grpdelay(b,a,128);
>> [h,f] = freqz(b,a,128,2000);
>> pd = -unwrap(angle(h))*(2000/(2*pi))./f;
>> plot(f,gd,'-',f,pd,'--')
>> axis([0 1000 -30 30])
>> legend('Group Delay','Phase Delay')
```

Kết quả.



Hình 22.12

22.19 Phân tích điểm cực – không

Hàm `zplane` vẽ các điểm cực và điểm không của một hệ thống tuyến tính. Ví dụ: cho một bộ lọc đơn giản với một điểm không tại $-1/2$ và một cặp điểm cực phức tại $0.9e^{j2\pi(0.3)}$ và $0.9e^{-j2\pi(0.3)}$;

```
pol = 0.9*exp(j*2*pi*[-0.3 0.3]');
```

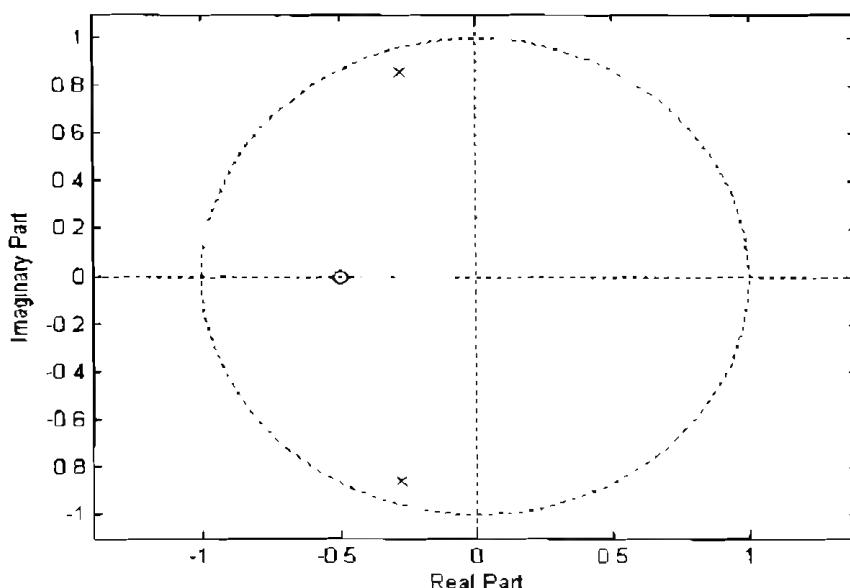
```
>> zer = -0.5;
```

```
>> pol = 0.9*exp(j*2*pi*[-0.3 0.3]');
```

Để vẽ các điểm cực – không của bộ lọc dùng lệnh `zplane(zer,pol)`:

```
>> zplane(zer,pol)
```

Kết quả:



Hình 22.13

Có thể chuyển các điểm cực và điểm không thành hàm truyền đạt với lệnh

```
>> [b,a] = zp2tf(zer,pol,1);
```

b , a là các vector hệ số của bộ lọc, sau đó ta có thể dùng hàm `fvtool` để hiển thị các đường đặc trưng khác. Hàm `zplane` cũng nhận các tham số ở dạng hàm truyền đạt `[b,a]`. `zplane(b,a)` Trong trường hợp này, hàm `zplane` trước hết tìm các điểm cực là các nghiệm của mẫu số và các điểm không là các nghiệm của tử số, sau đó hàm vẽ các điểm cực – không theo các giá trị này.

22.20 Phân tích Fourier rời rạc

Phân tích Fourier rời rạc viết tắt là DFT, là công cụ chủ yếu của xử lý số tín hiệu. Cơ sở của hộp công cụ là biến đổi Fourier nhanh (FFT). Nhiều hàm chức năng của hộp công cụ có dùng đến FFT như: đáp ứng tần số, phân tích phổ và một số hàm lọc. MATLAB cung cấp các hàm `fft` và `ifft` để tính biến đổi Fourier và biến đổi Fourier ngược. Nếu x là

một vector, fft tính DFT của vector đó, nếu x là mảng chữ nhật, fft tính DFT cho mỗi cột của mảng. Ví dụ:

Tạo một vector thời gian

```
>> t = (0:1/100:10-1/100);
```

Tạo tín hiệu

```
>> x = sin(2*pi*15*t) + sin(2*pi*40*t);
```

Tính DFT cho chuỗi biến đổi pha và biên độ của tín hiệu.

```
>> y = fft(x); % Tính DFT của x
```

```
>> m = abs(y); % Lấy giá trị biến đổi biên độ
```

```
>> p = unwrap(angle(y)); % Lấy giá trị biến đổi pha
```

Để vẽ biên độ và pha dùng các lệnh:

```
>> f = (0:length(y)-1)*99/length(y); % Frequency vector
```

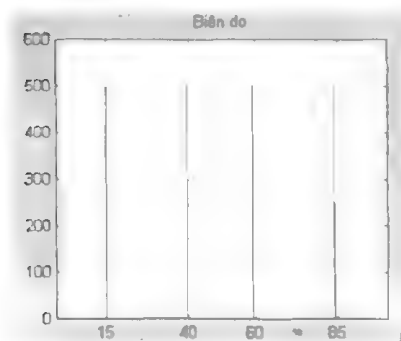
```
>> plot(f,m); title('Biên độ');
```

```
>> set(gca,'XTick',[15 40 60 85]);
```

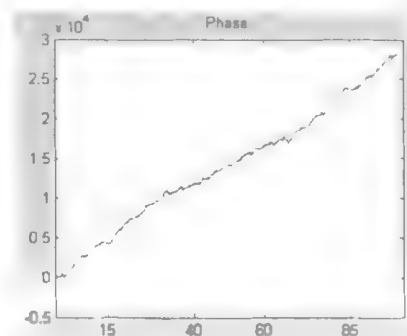
```
>> figure; plot(f,p*180/pi); title('Phase');
```

```
>> set(gca,'XTick',[15 40 60 85]);
```

Kết quả:



Hình 22.14a



Hình 22.14b

Tham số thứ hai của hàm fft là số điểm biến đổi fft. Gọi hàm $y = \text{fft}(x, n)$ sẽ biến đổi DFT x tại n điểm cách đều nhau trên vòng tròn đơn vị.

22.21 Mô hình hệ thống tuyến tính rời rạc

Hộp công cụ cung cấp một số mô hình để biểu diễn các hệ thống tuyến tính trong miền thời gian. Các tiện ích này giúp bạn chọn lựa được mô hình phù hợp nhất và cho

phép chuyển đổi giữa các mô hình một cách dễ dàng. MATLAB cung cấp hai dạng mô hình: rời rạc và liên tục. Hệ thống rời rạc thời gian có thể biểu diễn cho các bộ lọc số. Ở đây ta xem xét một số mô hình rời rạc trong MATLAB: hàm truyền đạt, biểu diễn cực – không - hệ số khuếch đại, không gian trạng thái, khai triển phân thức tối giản, tích phân thức bậc hai, cấu trúc Lattice và ma trận tích chập.

Hàm truyền đạt biểu diễn bộ lọc trong miền z và thể hiện như một tỷ số của hai đa thức. Mô hình hàm truyền đạt sử dụng các vector hệ số b và a để biểu diễn tử số và mẫu số.

Mô hình cực – không - hệ số khuếch đại là một biểu diễn khác của hàm truyền đạt. Các điểm cực là nghiệm của đa thức mẫu số và các điểm không là nghiệm của đa thức tử số, hệ số khuếch đại là một hằng số. Sử dụng hàm `roots` để tìm các nghiệm. Ví dụ, cho bộ lọc IIR có $b = [2 \ 3 \ 4]$; $a = [1 \ 3 \ 3 \ 1]$;

```
>> b = [2 3 4]; a = [1 3 3 1];
```

Các điểm không, điểm cực và hệ số khuếch đại của bộ lọc là:

```
>> q = roots(b)
```

```
q =
```

```
-0.7500 + 1.1990i
```

```
0.7500 - 1.1990i
```

```
p = roots(a)
```

```
p =
```

```
-1.0000
```

```
-1.0000 + 0.0000i
```

```
-1.0000 - 0.0000i
```

```
k = b(1)/a(1)
```

```
k =
```

```
2
```

Để chuyển đổi từ các điểm cực và điểm không về biểu diễn của hàm truyền đạt ta dùng lệnh

```
>> bb = k*poly(q)
```

```
bb =
```

```
2.0000 3.0000 4.0000
```

```
aa = poly(p)
```

```
aa =
```


1.0000 3.0000 3.0000 1.0000

Không gian trạng thái biểu diễn một bộ lọc số bởi một hệ các phương trình vi phân, hoặc một tập các phương trình vi phân bậc nhất.

Khai triển phân thức tối giản là biểu diễn của hàm truyền đạt ở dạng tổng của các phân thức tối giản có dạng:

$$\frac{b(z)}{a(z)} = \frac{r(1)}{1 - p(1)z^{-1}} + \dots + \frac{r(n)}{1 - p(n)z^{-1}} + k(1) + k(2)z^{-1} + \dots + k(m - n + 1)z^{-(m-n)}$$

hoặc nếu $a(z)$ có nghiệm bội p bậc s thì:

$$\frac{r(j)}{1 - p(j)z^{-1}} + \frac{r(j+1)}{(1 - p(j)z^{-1})^2} + \dots + \frac{r(j+s-1)}{(1 - p(j)z^{-1})^s}$$

Khai triển phân thức tối giản trong MATLAB bằng hàm `residuez` và thường dùng để thực hiện biến đổi z ngược. Ví dụ, cho hàm truyền đạt có $b = [-4 \ 8]$ và $a = [1 \ 6 \ 8]$:

```
>> b = [-4 8] và a = [1 6 8];
```

```
>> [r,p,k] = residuez(b,a)
```

```
r =
```

```
-12
```

```
8
```

```
p =
```

```
-4
```

```
-2
```

```
k =
```

```
[]
```

r là vector chứa các tử số của phân thức tối giản, p là các nghiệm của mẫu số, k là các hệ số của

z^i trong đó i thay đổi từ 0 đến $m-n$, trong đó m là bậc của tử số và n là bậc của mẫu số

Tích các phân thức bậc hai (SOS) là một cách biểu diễn của hàm truyền đạt:

$$H(z) = \prod_{k=1}^L H_k(z) = \prod_{k=1}^L \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{a_{0k} + a_{1k}z^{-1} + a_{2k}z^{-2}}$$

Trong đó L là số nhân tử của hệ thống. MATLAB thể hiện mỗi nhân tử bởi một mảng 6 phần tử, mỗi 3 phần tử đầu là các hệ số của tử số, ba phần tử sau là các hệ số của mẫu số. Thể hiện của toàn bộ các hệ số có dạng:

$$\varepsilon_{OS} = \begin{bmatrix} b_{01} & b_{11} & b_{21} & a_{01} & a_{11} & a_{21} \\ b_{02} & b_{12} & b_{22} & a_{02} & a_{12} & a_{22} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{0L} & b_{1L} & b_{2L} & a_{0L} & a_{1L} & a_{2L} \end{bmatrix}$$

Trong đó SOS là viết tắt của Second-Order Sections. Hàm `zp2sos` và `ss2sos` chuyển đổi biểu diễn hệ thống bởi các điểm cực – không và không gian trạng thái thành biểu diễn SOS.

22.22 Danh sách các hàm của hộp công cụ xử lý tín hiệu

Các hàm phân tích bộ lọc

<code>abs</code>	Lấy biên độ của tín hiệu
<code>angle</code>	Lấy góc pha của tín hiệu
<code>freqs</code>	Đáp ứng tần số của các bộ lọc tương tự
<code>freqspace</code>	Tạo khoảng tần số đáp ứng tần số
<code>freqz</code>	Đáp ứng tần số của các bộ lọc số
<code>fvtool</code>	Mở công cụ biểu diễn hình ảnh bộ lọc
<code>grpdelay</code>	Tính trễ nhóm
<code>impz</code>	Tính đáp ứng xung của bộ lọc số
<code>phasedelay</code>	Tính trễ pha của các bộ lọc số
<code>phasez</code>	Tính đáp ứng pha của các bộ lọc số
<code>stepz</code>	Tính đáp ứng xung của các bộ lọc số
<code>unwrap</code>	Làm trơn đường biểu diễn pha
<code>zerophase</code>	Tìm điểm pha không của các bộ lọc số
<code>zplane</code>	Vẽ điểm cực – không

Các hàm thực hiện lọc

<code>conv</code>	Thực hiện tích chập hai vector
<code>conv2</code>	Thực hiện tích chập hai chiều
<code>deconv</code>	Thực hiện giải chập
<code>fftfilt</code>	Tính FFT dựa trên lọc FIR dùng phương pháp overlap - add
<code>filter</code>	Thực hiện lọc IIR hoặc FIR
<code>filter</code>	Lọc 2 chiều

<code>filtfilt</code>	lọc số pha không
<code>filtic</code>	Tìm điều kiện đầu cho thực hiện lọc trực tiếp
<code>latcfilt</code>	Thực hiện lọc bậc thang – lattice
<code>medfilt1</code>	Lọc median một chiều
<code>sgolayfilt</code>	Lọc Savitzky-Golay

Thiết kế lọc số FIR

<code>convmtx</code>	Tích chập ma trận
<code>cremez</code>	Thiết kế lọc FIR pha phi tuyến
<code>dfilt</code>	Tạo bộ lọc dùng cú pháp hướng đối tượng
<code>fir1</code>	Thiết kế một bộ lọc đáp ứng xung hữu hạn dựa trên phương pháp cửa sổ
<code>fir1</code>	Thiết kế một bộ lọc FIR dựa trên phương pháp lấy mẫu tần số
<code>fircls</code>	Thiết kế bộ lọc FIR sai số bình phương tối thiểu cho lọc nhiều băng
<code>fircls</code>	Thiết kế bộ lọc FIR sai số bình phương tối thiểu cho lọc thông thấp và thông cao pha tuyến tính
<code>firgauss</code>	Thiết kế lọc FIR Gausse
<code>firls</code>	Thiết kế lọc FIR pha tuyến tính sai số bình phương tối thiểu
<code>firrcos</code>	Thiết kế lọc FIR cosin
<code>intfilt</code>	Thiết kế lọc FIR nội suy
<code>kaiserord</code>	Định lượng các tham số cho một thiết kế lọc FIR với cửa sổ Kaiser
<code>remez</code>	Tính toán tối ưu Parks-McClellan cho thiết kế bộ lọc FIR
<code>remez</code>	Tính toán bậc của bộ lọc tối ưu FIR Parks-McClellan
<code>sgolay</code>	Thiết kế bộ lọc Savitzky-Golay

Thiết kế bộ lọc số IIR

<code>butter</code>	Thiết kế lọc số và lọc tương tự Butterworth
<code>cheby1</code>	Thiết kế lọc Chebyshev loại 1 (gợn sóng tại giải thông)
<code>cheby2</code>	Thiết kế lọc Chebyshev loại 2 (gợn sóng tại giải chắn)
<code>ellip</code>	Thiết kế bộ lọc Elliptic (lọc Cauer)
<code>maxflat</code>	Thiết kế bộ lọc số Butterworth thông dụng

prony	Phương pháp Prony để thiết kế bộ lọc IIR trong miền thời gian
stmcb	Tính toán mô hình tuyến tính bằng lặp Steiglitz-McBride
yulewalk	Thiết kế lọc số hồi quy

Ước lượng bậc của bộ lọc IIR

buttord	Tính bậc và tần số cắt của bộ lọc Butterworth
cheb1ord	Tính bậc cho bộ lọc Chebyshev loại 1
cheb2ord	Tính bậc cho bộ lọc Chebyshev loại 2
ellipord	Tính bậc cực tiểu cho bộ lọc elliptic

Các dạng bộ lọc thông thấp tương tự

besselap	Bộ lọc thông thấp tương tự Bessel
buttap	Bộ lọc thông thấp tương tự Butterworth
cheb1ap	Bộ lọc thông thấp tương tự Chebyshev loại 1
cheb2ap	Bộ lọc thông thấp tương tự Chebyshev loại 2
ellipap	Bộ lọc thông thấp tương tự Elliptic

Thiết kế các bộ lọc tương tự

besself	Thiết kế bộ lọc tương tự Bessel
butter	Thiết kế bộ lọc số và bộ lọc tương tự Butterworth
cheby1	Thiết kế bộ lọc tương tự Chebyshev loại 1
cheby2	Thiết kế bộ lọc tương tự Chebyshev loại 2
ellip	Thiết kế bộ lọc Elliptic (Cauer)

Chuyển đổi các bộ lọc tương tự

lp2bp	Chuyển bộ lọc thông thấp thành thông dải
lp2bs	Chuyển bộ lọc thông thấp thành chặn dải
lp2hp	Chuyển bộ lọc thông thấp thành thông cao
lp2lp	Thay đổi tần số cắt cho bộ lọc thông thấp

Số hoá các bộ lọc tương tự

bilinear	Phương pháp chuyển đổi Bilinear
impinvar	Phương pháp bất biến xung

Chuyển đổi mô hình hệ thống tuyến tính

latc2tf	Chuyển đổi các tham số mô hình Lattice thành hàm truyền đạt
---------	---

polystab	Ổn định hoá hàm truyền đạt
polyscale	Tỷ lệ hoá các nghiệm của hàm truyền đạt
residuez	Khai triển hàm truyền đạt thành phân thức tối giản
sos2ss	Chuyển mô hình tích phân thức bậc hai thành dạng không gian trạng thái
sos2tf	Chuyển mô hình tích các phân thức bậc hai thành hàm truyền đạt
sos2zp	Chuyển mô hình tích các phân thức bậc hai thành dạng cực – không
ss2sos	Chuyển mô hình không gian trạng thái thành các tích phân thức bậc hai
ss2zp	Chuyển mô hình không gian trạng thái thành dạng cực – không
tf2latc	Chuyển hàm truyền đạt thành cấu trúc lọc Lattice
tf2sos	Chuyển hàm truyền đạt thành mô hình tích các phân thức bậc hai
tf2ss	Chuyển hàm truyền đạt thành mô hình tích các phân thức bậc hai
tf2zp	Chuyển hàm truyền đạt analog thành biểu diễn cực – không
tf2zpk	Chuyển hàm truyền đạt rời rạc thành biểu diễn cực – không
zp2ss	Chuyển biểu diễn cực – không thành dạng không gian trạng thái
zp2tf	Chuyển biểu diễn cực – không thành hàm truyền đạt

Các hàm cửa sổ

barthannwin	Tính các hệ số cửa sổ Bartlett-Hann nâng cấp
bartlet	Tính các hệ số cửa sổ Bartlett
blackman	Tính các hệ số cửa sổ Blackman
blackmanharris	Tính các hệ số cửa sổ tối thiểu 4 phần tử Blackman-Harris
bohmanwin	Tính cửa sổ Bohman
chebwin	Tính cửa sổ Chebyshev
flattopwin	Tính cửa sổ đỉnh phẳng
gausswin	Tính cửa sổ Gausse
hamming	Tính cửa sổ Hamming

hann	Tính cửa sổ Hann
kaiser	Tính cửa sổ Kaiser
nuttallwin	Tính cửa sổ tối thiểu Nuttall – 4 phần tử Blackman-Harris.
parzenwin	Tính cửa sổ Parzen
rectwin	Tính cửa sổ chữ nhật
sigwin	Tính cửa sổ chữ nhật bằng củ pháp hướng đối tượng
triang	Tính cửa sổ tam giác
tukeywin	Tính cửa sổ Tukey
window	Mở chương trình giao diện đồ hoạ tính toán các hệ số cửa sổ

Các hàm biến đổi tín hiệu

bitrevoder	Hoán vị các đầu vào theo thứ tự nghịch đảo bit
ctz	Biến đổi Chirp z
dct	Biến đổi co sin rời rạc (DCT)
dftmtx	Ma trận biến đổi Fournier rời rạc
fft	Tính FFT một chiều
fft2	Tính FFT hai chiều
fftshift	Sắp xếp lại các đầu ra của hàm FFT
goertzel	Tính toán biến đổi Fourier rời rạc dùng thuật toán Goertzel bậc hai
hilbert	Tính toán biến đổi thời gian rời rạc dùng biến đổi Hilbert
idct	Biến đổi cosin rời rạc ngược
ifft	Biến đổi ngược FFT một chiều
ifft2	Biến đổi ngược FFT hai chiều

Phân tích phổ và xử lý thống kê

cohere	Ước lượng liên kết bình phương biên độ giữa hai tín hiệu
corrcoef	Tính ma trận hệ số tương quan
cormtx	Tính một ma trận dữ liệu cho ước lượng ma trận tự tương quan
cov	Tính ma trận covarian
csd	Tính mật độ phổ chéo của hai tín hiệu
pburg	Tính mật độ phổ công suất bằng phương pháp Burg

pcov	Tính mật độ phổ công suất bằng phương pháp covariance
peig	Tính phổ giả bằng phương pháp vector riêng
periodogram	Tính mật độ phổ công suất (PSD) của tín hiệu
pmcov	Tính mật độ phổ công suất bằng phương pháp covarian thay đổi
pmtm	Tính mật độ phổ công suất bằng phương pháp MTM
pmusic	Tính mật độ phổ công suất bằng phương pháp MUSIC
psdplot	Vẽ các dữ liệu mật độ phổ công suất PSD
pwelch	Tính mật độ phổ công suất bằng phương pháp Welch
pyulear	Tính mật độ phổ công suất bằng phương pháp Yule-Walker AR
rooteig	Tính tần số và công suất bằng giải thuật nghiệm MUSIC
tfe	Tính hàm truyền đạt từ đầu vào và đầu ra
xcorr	Tính hàm tương quan chéo
xcorr2	Tính hàm tương quan chéo tín hiệu hai chiều
xcov	Tính hàm hiệp biến chéo

Mô hình tham số

arburg	Tính các tham số mô hình AR bằng phương pháp Burg
arcov	Tính các tham số mô hình AR bằng phương pháp covarian
armcov	Tính các tham số mô hình AR bằng phương pháp covarian thay đổi
invfreqs	Nhận dạng các tham số bộ lọc rời rạc từ các dữ liệu đáp ứng tần số

Dự đoán tuyến tính

ac2poly	Chuyển một chuỗi tự tương quan thành đa thức dự đoán
ac2rc	Chuyển một chuỗi tự tương quan thành các hệ số ảnh
is2rc	Chuyển các tham số sin nghịch đảo thành các hệ số phản xạ
lar2rc	Chuyển các tham số tỷ lệ loga thành các hệ số ảnh
levinson	Tính hồi quy Levinson-Durbin
lpc	Tính các hệ số của bộ lọc dự đoán tuyến tính
lsf2poly	Chuyển các tần số phổ đường thẳng thành các hệ số bộ lọc dự đoán
poly2ac	Chuyển một đa thức lọc dự đoán thành các hệ số ảnh

rc2ac	Chuyển các hệ số ảnh thành chuỗi tự tương quan
rc2is	Chuyển các hệ số ảnh thành các tham số sin ngược
rlevinson	Tính hồi quy Levinson-Durbin ngược
schurrc	Tính các hệ số ảnh từ chuỗi tự tương quan

Xử lý tín hiệu đa tốc độ

decimate	Giảm tốc độ lấy mẫu của một chuỗi
downsample	Giảm tốc độ lấy mẫu theo một hệ số nguyên
interp	Tăng tốc độ lấy mẫu theo một hệ số nguyên
interp1	Tăng tốc độ lấy mẫu theo một hệ số nguyên
resample	Thay đổi tốc độ lấy theo một hệ số bất kỳ
spline	Nội suy Cubic spline
upfirdn	Tăng tần số lấy mẫu, áp dụng cho bộ lọc FIR và giảm tần số lấy mẫu
upsample	Tăng tần số lấy mẫu bởi một hệ số nguyên

Tạo các dạng tín hiệu

chirp	Tạo tín hiệu cosin có tần số biến đổi tuyến tính
diric	Tính hàm sinc tuần hoàn hoặc Dirichlet
gauspuls	Tạo xung hình sin điều chế Gausse
gmonopuls	Tạo xung Gausse đơn
pulstran	Tạo chuỗi xung
rectpuls	Tạo xung chữ nhật
sawtooth	Tạo xung răng cưa hoặc tam giác
sinc	Tạo hàm sinc
square	Tạo xung vuông
tripuls	Tạo xung tam giác
vco	Tạo dao động điều khiển bởi điện áp

Các hàm đặc biệt

buffer	Đệm một vector tín hiệu vào một ma trận hoặc các khung dữ liệu
cell2sos	Chuyển một mảng phân tử vào ma trận các hệ số nhân tử bậc hai
cplxpair	Nhóm các số phức thành các cặp số phức liên hợp

demod	Giải điều chế cho trường hợp truyền thông
dpss	Chuỗi Slepian
dpsscLEAR	Loại bỏ chuỗi Slepian khỏi cơ sở dữ liệu
dpssdir	Thư mục cơ sở dữ liệu chuỗi Slepian
dpssload	Nhập chuỗi Slepian từ cơ sở dữ liệu
dpsssave	Lưu chuỗi Slepian vào cơ sở dữ liệu
eqtfLength	Tạo chiều dài của hàm truyền đạt
modulate	Điều chế trong trường hợp truyền thông
seqperiod	Tính chu kỳ của chuỗi
sos2cell	Chuyển ma trận hệ số sos thành mảng phân tử
specgram	Phân tích tần số - thời gian
stem	Vẽ chuỗi rời rạc
strips	Vẽ ma trận hoặc vector
udecode	Giải mã số nguyên lượng tử 2^n mức thành số dấu phẩy động đầu ra
uencode	Lượng tử và mã hoá số dấu phẩy động thành số nguyên đầu ra

Các giao diện đồ họa

fdatool	Mở công cụ phân tích và thiết kế bộ lọc
fvtool	Mở công cụ hiển thị đồ họa bộ lọc
spool	Công cụ xử lý tín hiệu số tương tác
wintool	Mở công cụ phân tích và thiết kế cửa sổ
wvtool	Mở công cụ hiển thị đồ họa cửa sổ

HỘP CÔNG CỤ TRUYỀN THÔNG

23.1 Nhiều Gauss trắng

Hàm `wgn` tạo ra các ma trận ngẫu nhiên sử dụng một phân bố nhiễu Gauss trắng. Ví dụ dòng lệnh sau sẽ tạo ra một vector cột chiều dài 50 chứa nhiễu Gauss trắng thực mà công suất là 2 dBW. Hàm sử dụng giả thiết trở kháng tải là 1 ohm.

```
>> y1 = wgn(50,1,2)
```

Để tạo ra nhiễu trắng Gauss phức với công suất 2 Watt, trên một tải 60 ohm, sử dụng một trong các dòng lệnh dưới đây. Chú ý rằng thứ tự của các chuỗi vào không ảnh hưởng đến kết quả tính toán.

```
>> y2 = wgn(50,1,2,60,'complex','linear');
```

```
>> y3 = wgn(50,1,2,60,'linear','complex');
```

Để gửi một tín hiệu qua một kênh nhiễu Gauss trắng cộng tính, sử dụng hàm `awgn`.

23.2 Các ma trận symbol ngẫu nhiên

Hàm `randsrc` tạo ra các ma trận ngẫu nhiên với đầu vào được chọn một cách độc lập từ bảng chữ cái mà bạn chỉ ra, với một sự phân bố xác suất cũng do bạn xác định. Ví dụ, dòng lệnh dưới đây tạo ra một ma trận 5*4 có các đầu vào được chọn độc lập và phân phối đồng đều trong tập {1,3,5} (Kết quả thực tế của bạn có thể thay đổi bởi vì nó là ngẫu nhiên).

```
>> a = randsrc(5,4,[1,3,5])
```

```
a =
```

```
3    5    1    5
```

```
1    5    3    3
```

```
1    3    3    1
```

```
1    1    3    5
```

```
3    1    1    3
```

Nếu bạn muốn 1 xuất hiện nhiều cỡ gấp 2 lần 3 hoặc 5, thì sử dụng dòng lệnh bên dưới để quy định sự phân bố lệch. Chú ý rằng đối số vào thứ 3 gồm hai hàng, một chỉ thị các giá trị có thể của b và cái còn lại chỉ báo xác suất của mỗi giá trị.

```
b = randsrc(5,4,[1,3,5; .5,.25,.25])
```

b =

3	3	5	1
1	1	1	1
1	5	1	1
1	3	1	3
3	1	3	1

23.3 Tạo các ma trận số nguyên ngẫu nhiên

Hàm `randint` tạo ra các ma trận số nguyên ngẫu nhiên có các đầu vào nằm trong một dải do bạn chỉ ra. Một trường hợp đặc biệt tạo ra ma trận nhị phân ngẫu nhiên. Ví dụ, lệnh dưới đây tạo ra một ma trận 5*4 chứa các số nguyên ngẫu nhiên nằm giữa 2 và 10.

```
>> c = randint(5,4,[2,10])
```

c =

2	4	4	6
4	5	10	5
9	7	10	8
5	5	2	3
10	3	4	10

Nếu dải chọn của bạn là [0,10] thay vì [2,10] thì bạn có thể sử dụng một trong các lệnh dưới đây. Chúng tạo ra các kết quả số khác nhau, nhưng sử dụng cùng sự phân phối.

```
>> d = randint(5,4,[0,10]);
```

```
>> e = randint(5,4,11);
```

23.4 Các sơ đồ lỗi bit ngẫu nhiên

Hàm `randerr` tạo ra các ma trận có các đầu vào hoặc là 0 hay 1. Tuy nhiên, các tùy chọn của nó là hơi khác với các tùy chọn của `randint`, bởi vì `randerr` là phương tiện để

kiểm tra mã hóa điều khiển lỗi. Ví dụ, lệnh dưới đây tạo ra một ma trận 5×4 có các thuộc tính mà mỗi hàng chứa chính xác một giá trị 1.

```
f = randerr(5,4)
```

```
f =
```

```
0  0  1  0
0  0  1  0
0  1  0  0
1  0  0  0
0  0  1  0
```

Bạn có thể sử dụng một lệnh để làm rối một mã nhị phân mà chứa 5 từ mã 4 bit. Thêm ma trận f và ma trận mã của bạn (modulo 2) sẽ tạo ra chính xác một lỗi trong mỗi từ mã. Nói cách khác, nếu muốn làm xáo trộn mỗi từ mã nhờ tạo ra một lỗi với xác suất 0.4 và hai lỗi với xác suất 0.6, thì bạn có thể sử dụng lệnh sau.

```
% Mỗi hàng có một giá trị '1' với xác suất 0.4, còn lại là hai giá trị '1'
```

```
>>g = randerr(5,4,[1,2, 0.4,0.6])
```

```
g =
```

```
0  1  1  0
0  1  0  0
0  0  1  1
1  0  1  0
0  1  1  0
```

Chú ý: Ma trận xác suất là đối số thứ ba của `randerr` chỉ ảnh hưởng đến số các giá trị 1 ở mỗi dòng, chứ không phải vị trí đặt nó

Trong ứng dụng khác, bạn có thể sử dụng một trong các lệnh sau để tạo ra vector cột 100 phần tử nhị phân đồng xác suất. Ba lệnh tạo ra các đầu ra số khác nhau, nhưng sử dụng cùng một phân phối. Ta thấy rằng các đối số đầu vào thứ 3 thay đổi tùy theo mỗi cách đặc tả ứng xử của nó nhất định của mỗi hàm.

```
>> binarymatrix1 = randsrc(100,1,[0 1]); % Các giá trị có thể là 0,1.
```

```
>> binarymatrix2 = randint(100,1,2); % Hai giá trị có thể
```

```
>> binarymatrix3 = randerr(100,1,[0 1;.5 .5]); % Không có giá trị 1 hoặc là 1 giá trị 1
```

23.5 Tỷ số lỗi

Tỷ số lỗi là một chỉ số đánh giá mức độ chính xác của hệ thống truyền tin, được tính bởi số bit lỗi trên tổng số bit truyền. So sánh các bản tin trước và sau khi truyền có thể

giúp bạn đánh giá chất lượng của việc thiết kế hệ thống thông tin hoặc hiệu quả của một kỹ thuật hay các thuật toán chống lỗi đặc biệt. Nếu hệ thống thông tin của bạn sử dụng một số bit để đại diện cho một symbol đơn thì việc đếm lỗi bit có khác với việc đếm lỗi symbol. Trong trường hợp đếm bit hay là symbol, tỉ số lỗi là số các lỗi chia cho tổng số (bit hoặc symbol) được truyền.

Hàm `biterr` so sánh hai bản tin và tính số các lỗi bit và tỉ số lỗi bit. Hàm `symerr` so sánh hai bản tin và tính số các lỗi symbol và tỉ số lỗi symbol. Đoạn chương trình dưới đây sử dụng hàm `symerr` để tính tỉ số lỗi symbol cho mã khối tuyến tính có nhiễu. Sau khi thêm nhiễu vào thông điệp đã mã hóa, nó so sánh mã có nhiễu kết quả với mã gốc. Sau đó nó giải mã và so sánh bản tin được giải mã với bản tin gốc. Bạn có thể có gổ đoạn chương trình này vào m file và thực hiện lệnh `run` (F5):

```
m = 3; n = 2^m-1, k = n-m; % Chuẩn bị sử dụng mã Hamming.
msg = randint(k*200,1,2), % 200 bản tin k bit.
code = encode(msg,n,k,'hamming');
codenoisy = rem(code+(rand(n*200,1)>.95),2); % Thêm nhiễu.
% Giải mã và sửa vài lỗi.
newmsg = decode(codenoisy,n,k,'hamming');
% Tính và hiển thị tỉ số lỗi symbol.
[codenum,coderate] = symerr(code,codenoisy);
[msgnum,msgrate] = symerr(msg,newmsg);
disp(['Error rate in the received code: ',num2str(coderate)])
disp(['Error rate after decoding: ',num2str(msgrate)])
```

Kết quả hiển trong cửa sổ Command Window.

```
>> Error rate in the received code: 0.062857
```

```
Error rate after decoding: 0.0275
```

Kết quả trên cho thấy tỉ số lỗi giảm sau khi giải mã bởi vì bộ giải mã Hamming đã sửa một số lỗi. Kết quả của bạn có thể khác bởi vì ví dụ sử dụng các số ngẫu nhiên

23.6 So sánh tỉ số lỗi symbol và tỉ số lỗi bit

Trong ví dụ ở trên, lỗi symbol và lỗi bit là giống nhau bởi vì mỗi symbol là một bit. Trong thực tế một symbol có thể biểu diễn bởi nhiều bit. Các lệnh dưới đây thể hiện sự khác nhau giữa lỗi symbol và lỗi bit trong các tình huống khác

```
>> a = [1 2 3]; b = [1 4 4];
```

```
>> format rat % Hiển thị theo phân số thay cho thập phân.
```

```
>> [snum,srate] = symerr(a,b)
```

```
snum =
```

```
2
```

```
srate =
```

```
2/3
```

```
>> [bnum,brate] = biterr(a,b)
```

```
bnum =
```

```
5
```

```
brate =
```

```
5/9
```

bnum là 5 bởi vì giá trị vào thứ hai khác nhau hai bit và giá trị thứ 3 khác nhau 3 bit. brate là 5/9 vì tổng số các bit là 9. Tổng số các bit theo định nghĩa là số các giá trị đầu vào gấp a hoặc b lần số tối đa các bit giữa các giá trị vào của a và b.

23.7 Cộng nhiễu trắng Gaussian vào tín hiệu

Lệnh `y = awgn(x,snr)` sẽ cộng nhiễu trắng Gaussian vào vector tín hiệu `x`. Số vô hướng `snr` chỉ ra tỉ số tín hiệu/tạp âm (S/N) tính theo db. Nếu `x` là phức thì `awgn` cộng nhiễu phức. Cú pháp này xem rằng công suất của `x` là 0 dBW

`y = awgn(x,snr,sigpower)` giống với cú pháp ở trên, ngoại trừ `sigpower` là công suất của `x` tính theo dBW.

`y = awgn(x,snr,'measured')` giống với `y = awgn(x,snr)`, ngoại trừ `awgn` thực hiện đo công suất của `x` trước khi cộng nhiễu.

`y = awgn(x,snr,sigpower,state)` giống với `y = awgn(x,snr,sigpower)`, ngoại trừ rằng `awgn` trước tiên khởi động lại trạng thái của bộ phát số ngẫu nhiên `randn` về trạng thái số nguyên.

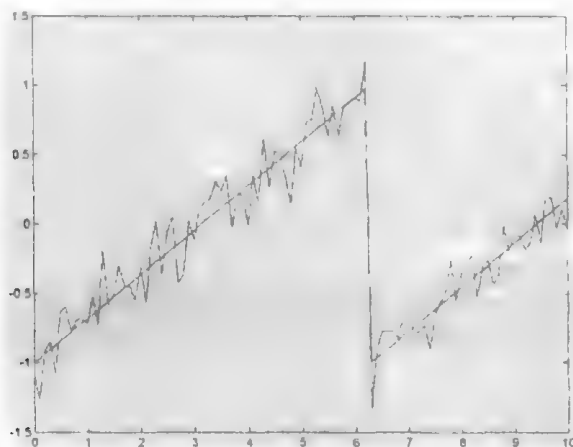
`y = awgn(...,powertype)`, giống với các cú pháp ở trên, ngoại trừ rằng chuỗi `powertype` chỉ ra đơn vị của `snr` và `sigpower`. Các lựa chọn cho `powertype` là 'db' và 'linear'. Nếu `powertype` là 'db', thì `snr` được tính bằng dB và `sigpower` được tính bằng dBW. Nếu `powertype` là 'linear', thì `snr` được tính bằng số lần và `sigpower` được tính bằng W.

Ví dụ Các lệnh dưới đây cộng nhiễu trắng Gaussian vào tín hiệu xung răng cưa. Sau đó vẽ tín hiệu ban đầu và tín hiệu có nhiễu.

```
t = 0:1:10;
```

```
x = sawtooth(t); % Tạo ra tín hiệu xung răng cưa
```

`y = awgn(x,10,'measured');` % Cộng nhiễu trắng Gaussian
`plot(t,x,t,y)` % Vẽ cả 2 tín hiệu.

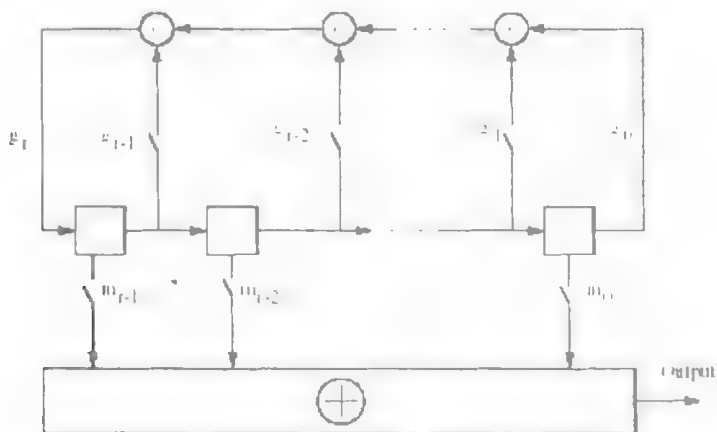


Hình 23.1

Xem thêm `wgn`, `randn`.

23.8 Các bộ sinh chuỗi giả nhiễu, mặt nạ và giá trị dịch

Các thanh ghi dịch phản hồi tuyến tính là một phần của bộ sinh chuỗi giả nhiễu. Dưới đây là đồ thị hình vẽ của bộ sinh chuỗi giả nhiễu. Tất cả các bộ công thực hiện phép cộng modulo 2.



Hình 23.2

Đa thức gốc xác định trạng thái của mỗi khóa có nhãn g_i trong đó mặt nạ xác định trạng thái của mỗi khóa có tên m_i . Dưới đây là đồ thị cho thấy cách thực hiện dịch,

mà làm trễ điểm bắt đầu của chuỗi đầu ra. Nếu giá trị dịch là 0 thì khóa m_0 được đóng trong khi các khóa m_k khác mở. Bảng dưới đây chỉ ra giá trị dịch ảnh hưởng đến đầu ra thành ghi dịch như thế nào.

	$T = 0$	$T = 1$	$T = 2$...	$T = s$	$T = s+1$
Shift = 0	x_0	x_1	x_2	...	x_s	x_{s+1}
Shift = $s > 0$	x_s	x_{s+1}	x_{s+2}	...	x_{2s}	x_{2s+1}

Nếu muốn tạo ra chuỗi giả nhiễu trong Simulink, xem tài liệu tham khảo cho khối PN Sequence Generator.

Ví dụ Lệnh dưới đây chuyển đổi giá trị dịch 5 thành mặt nạ tương đương x^2+x+1 cho thành ghi dịch phản hồi tuyến tính mà các đường nối của nó được chỉ ra bởi đa thức nguyên tố $x^4 + x^3 + 1$.

```
>> mk = shift2mask([1 1 0 0 1],5)
```

```
mk =
```

```
1 0 1 1
```

Xem thêm mask2shift, deconv, isprimitive, primpoly.

23.9 Mã hóa/giải mã BCH

Hàm `msg = bchdeco(code,k,t)` thực hiện giải mã sử dụng phương pháp BCH. Trong đó k là chiều dài bản tin. Chiều dài từ mã n phải ở dạng 2^m-1 với số nguyên m lớn hơn hoặc bằng 3. Giá trị `code` là 1 ma trận nhị phân với n cột, mỗi hàng biểu diễn 1 từ mã. Hàm trả lại ma trận `msg` là 1 ma trận nhị phân với k cột, mỗi hàng biểu diễn 1 bản tin. t là khả năng sửa lỗi. Giải mã BCH đòi hỏi 1 đa thức gốc cho $GF(2^m)$, cú pháp này sử dụng đa thức gốc mặc định, `gfprimdf(m)`.

`msg = bchdeco(code,k,t,prim_poly)` giống với cú pháp ở trên, ngoại trừ rằng `prim_poly` là 1 vector hàng mà chứa các hệ số theo thứ tự bậc tăng của đa thức gốc cho $GF(2^m)$ mà sẽ được sử dụng trong suốt quá trình xử lý.

`[msg,err] = bchdeco()` trả về vector cột `err` mà cung cấp thông tin về sửa lỗi. 1 số nguyên không âm trong `err(r)` chỉ ra số lỗi được sửa trong từ mã thứ r ; 1 số nguyên âm chỉ ra rằng có nhiều lỗi hơn trong từ mã thứ r mà có thể sửa được.

`[msg,err,ccode] = bchdeco(...)` trả về mã được sửa trong `ccode`.

Ví dụ: Đoạn mã dưới đây mã hóa 1 bản tin ngẫu nhiên, mô phỏng sự cộng nhiễu vào mã, và sau đó giải mã bản tin.

```
m = 4, n = 2^m-1; % Chiều dài từ mã
```

```
params = bchpoly(n),
```



```

% Arbitrarily focus on 3rd row of params.
k = params(3,2); % Codeword length
t = params(3,3); % Khả năng sửa lỗi
msg = randint(100,k);
code = bchenco(msg,n,k); % Mã hóa bản tin
% Gây nhiễu t bit trong mỗi từ mã
noisycode = rem(code + randerr(100,n,1:t),2);
% Giải mã mã bị nhiễu
[newmsg,err,ccode] = bchdeco(noisycode,k,t);
if ccode==code
disp('All errors were corrected.')
end
if newmsg==msg
disp('The message was recovered perfectly.')
end

```

Trong trường hợp này, tất cả các lỗi được sửa và bản tin được khôi phục hoàn hảo. Tuy nhiên, nếu dòng thứ 9 được thay bằng:

```
noisycode = rem(code + randerr(100,n,1:(t+1)),2);
```

thì một số từ mã sẽ có nhiều hơn t lỗi. Do quá nhiều lỗi nên giải mã sẽ không chính xác.

23.10 Bộ giải mã khối

Hàm `msg = decode(code,n,k,'hamming/fmt',prim_poly)` thực hiện mã hóa sử dụng phương pháp Hamming. Cho cú pháp này, n phải ở dạng $2^m - 1$ cho các số nguyên m lớn hơn hoặc bằng 3, và k phải bằng $n - m$. Vec tơ `prim_poly` là một vector hàng để đưa ra các hệ số nhị phân theo thứ tự bậc tăng của đa thức nguyên tố cho $GF(2^m)$ được sử dụng trong quá trình mã hóa. Giá trị mặc định của `prim_poly` là `gfprimdf(m)`. Bảng mã hóa mã hàm sử dụng để sửa một lỗi đơn trong mỗi từ mã là `syndtable(hammgen(m))`.

Hàm `msg = decode(code,n,k,'linear/fmt',genmat,trt)` thực hiện giải mã, trong đó 1 mã khối tuyến tính được xác định bởi ma trận sinh $k \times n$ `genmat`. `genmat` được yêu cầu ở đầu vào. `decode` thực hiện sửa lỗi sử dụng bảng giải mã `trt`, trong đó `trt` là 1 ma trận $2^{(n-k)} \times n$.

Hàm `msg = decode(code,n,k,'cyclic/fmt',genpoly,trt)` thực hiện giải mã mã vòng và thực hiện sửa lỗi sử dụng bảng giải mã `trt`, trong đó `trt` là ma trận $2^{(n-k)} \times n$. `genpoly` là 1 vector hàng mã đưa ra các hệ số theo thứ tự bậc tăng của đa thức sinh nhị phân của mã.

Giá trị mặc định của `genpoly` là `cyclpoly(n,k)`. Theo định nghĩa, đa thức sinh cho 1 mã vòng $[n,k]$ phải có bậc $n-k$ và phải chia x^n-1 .

Hàm `msg = decode(code,n,k,'bch/fm',t,prim_poly)` thực hiện giải mã sử dụng phương pháp BCH. `prim_poly` là vector hàng mã đưa ra các hệ số theo thứ tự tăng của bậc của đa thức nguyên tố cho $GF(2^m)$ mà sẽ được sử dụng trong suốt quá trình xử lý. Giá trị mặc định của `prim_poly` là `gprimdf(m)`. Cho cú pháp này, n phải có dạng 2^m-1 cho các số nguyên m lớn hơn hoặc bằng 3. k và t phải là 1 chiều dài bản tin hợp lệ và khả năng sửa lỗi, như được đưa ra trong cột thứ 2 và 3 của 1 hàng của `params`:

```
params = bchpoly(n)
```

```
msg = decode(code,n,k) giống với msg = decode(code,n,k,'hamming/binary').
```

Hàm `[msg,err] = decode(...)` trả về 1 vector cột `err` mà đưa ra thông tin về sửa lỗi. Nếu mã là 1 mã chập thì `err` có các tính toán được sử dụng trong quá trình quyết định giải mã. Cho các loại mã khác, 1 số nguyên không âm trong hàng thứ r của `err` (hay hàng thứ r của `vec2mat(err,k)` nếu mã là 1 vector cột) chỉ ra số lỗi được sửa trong từ bản tin thứ r ; 1 số nguyên âm chỉ ra rằng có nhiều lỗi trong từ thứ r hơn khả năng có thể sửa.

```
[msg,err,ccode] = decode(...) trả về mã được sửa trong ccode.
```

`[msg,err,ccode,cerr] = decode(...)` trả về 1 vector cột `cerr` mà ý nghĩa của nó phụ thuộc vào dạng của mã:

- Nếu mã là 1 vector nhị phân, thì 1 số nguyên không âm trong hàng thứ r của `vec2mat(cerr,n)` chỉ ra số lỗi được sửa trong từ mã thứ r ; 1 số nguyên âm chỉ ra rằng có nhiều lỗi trong từ mã thứ r hơn khả năng có thể sửa được.

- Nếu mã không phải là 1 vector nhị phân, thì `cerr = err`.

Vi dụ: Trong trang tham chiếu cho `encode`, một số ví dụ minh họa cách sử dụng hàm `decode`

Vi dụ dưới đây minh họa cách sử dụng `err` và `cerr` khi phương pháp mã hóa không phải là mã chập và mã là 1 vector nhị phân. Đoạn mã thực hiện mã hóa 2 bản tin 5 bit sử dụng mã BCH. Mỗi từ mã có 15 bit. Các khối được cộng vào 2 bit của từ mã đầu tiên và bit đầu tiên của từ mã thứ 2. Sau đó `decode` được sử dụng để khôi phục bản tin ban đầu. Kết quả là các lỗi được sửa. `err` có cùng kích thước với `msg` và `cerr` có cùng kích thước với `code`. `err` phản ánh sự thực rằng bản tin đầu tiên được khôi phục sau khi sửa 2 lỗi, trong khi bản tin thứ 2 được khôi phục sau khi sửa 1 lỗi. `cerr` phản ánh sự thực rằng từ mã đầu tiên được giải mã sau khi sửa 2 lỗi, trong khi từ mã thứ 2 được giải mã sau khi sửa một lỗi.

```
m = 4; n = 2^m-1; % Chiều dài từ mã là 15.
```

```
k = 5; % Chiều dài bản tin hợp lệ cho mã BCH khi n = 15
```

```
t = 3, % Khả năng sửa lỗi tương ứng
```

```
msg = ones(10,1); % 2 bản tin, 5 bit cho mỗi bản tin
```

```
code = encode(msg,n,k,'bch');    % Mã hóa bản tin.
% Thay 2 lỗi vào từ đầu tiên và một lỗi và từ thứ 2 Tạo ra các lỗi này bằng cách %
đảo bit.
```

```
noisycode = code;
noisycode(1:2) = bitxor(noisycode(1:2),[1 1]);
noisycode(16) = bitxor(noisycode(16),1);
% Giải mã và sửa lỗi.
[newmsg,err,ccode,cerr] = decode(noisycode,n,k,'bch',t),
disp('Transpose of err is'); disp(err')
disp('Transpose of cerr is'); disp(cerr')
```

Đầu ra như dưới đây.

```
Transpose of err is
2 2 2 2 2 1 1 1 1 1 1
Transpose of cerr is
Columns 1 through 12
2 2 2 2 2 2 2 2 2 2 2 2
Columns 13 through 24
2 2 2 1 1 1 1 1 1 1 1 1
Columns 25 through 30
1 1 1 1 1 1
```

Thuật toán Phụ thuộc vào phương pháp giải mã. decode dựa vào các hàm cấp thấp như hamngen, syndtable, cyc gen, and bondeco.

23.11 Bộ giải mã khối

Hàm `code = encode(msg,n,k,'linear/fmt',genmat)` thực hiện mã hóa msg sử dụng genmat như là ma trận sinh cho phương pháp mã hóa khối tuyến tính, 1 ma trận kích thước $k \times n$ được yêu cầu ở đầu vào.

Hàm `code = encode(msg,n,k,'cyclic/fmt',genpoly)` mã hóa msg và tạo ra 1 mã vòng hệ thống genpoly là 1 vector hàng mà đưa ra các hệ số theo thứ tự bậc tăng của đa thức sinh nhị phân. Giá trị mặc định của genpoly là `cycpoly(n,k)`. Theo định nghĩa đa thức sinh cho 1 mã vòng $[n,k]$ phải có bậc $n-k$ và phải chia cho x^n-1 .

Hàm `code = encode(msg,n,k,'bch/fmt',genpoly)` mã hóa msg sử dụng phương pháp mã hóa BCH. genpoly là vector hàng mà đưa ra các hệ số theo thứ tự bậc tăng của đa

thức sinh BCH nhị phân bậc $-(n-k)$. Giá trị mặc định của `genpoly` là `bchpoly(n,k)`. Đối với cú pháp này, n phải có dạng 2^m-1 với số nguyên m lớn hơn hoặc bằng 3. Giá trị k phải là chiều dài bản tin hợp lệ như được đưa ra trong cột thứ hai của `params` trong lệnh `params = bchpoly(n)`.

Hàm `code = encode(msg,n,k,'hamming/fmt',prim_poly)` mã hóa `msg` sử dụng phương pháp mã hóa Hamming. Trong cú pháp này, n phải có dạng 2^m-1 với số nguyên m lớn hơn hoặc bằng 3, và k phải bằng $n-m$. Vec tơ `prim_poly` là 1 vector hàng mà đưa ra các hệ số nhị phân theo thứ tự bậc tăng của đa thức nguyên tố cho $GF(2^m)$ mà được sử dụng trong quá trình mã hóa. Giá trị mặc định của `prim_poly` là đa thức nguyên tố `gfprimdf(m)`.

Hàm `code = encode(msg,n,k)` giống với `code = encode(msg,n,k,'hamming/binary')`.

Hàm `[code,added] = encode(...)` trả về biến phụ `added`. `added` là số lỗi mà được đặt ở cuối của ma trận bản tin trước khi mã hóa để ma trận có kích thước phù hợp. Sự phụ hợp phụ thuộc vào n , k , kích thước của `msg`, và phương pháp mã hóa.

Ví dụ Ví dụ dưới đây minh họa 3 định dạng thông tin khác nhau (vector nhị phân, ma trận nhị phân, và vector thập phân) cho mã Hamming 3 bản tin có nội dung giống nhau trong các định dạng khác nhau; kết quả là 3 mã mà `encode` tạo ra có nội dung giống trong các định dạng khác.

```
m = 4, n = 2^m-1, % Chiều dài từ mã là 15
k = 11; % Chiều dài bản tin
% Tạo ra 100 bản tin, mỗi bản tin có k bit.
msg1 = randint(100*k,1,[0,1]); % Là 1 vector cột
msg2 = vec2mat(msg1,k), % Là 1 ma trận có k cột
msg3 = bi2de(msg2); % Là 1 cột các số nguyên
% Tạo ra 100 từ mã, mỗi từ mã có n bit.
code1 = encode(msg1,n,k,'hamming/binary');
code2 = encode(msg2,n,k,'hamming/binary');
code3 = encode(msg3,n,k,'hamming/decimal');
if ( vec2mat(code1,n)==code2 & de2bi(code3,n)==code2 )
    disp('All three formats produced the same content.')
end
```

Kết quả trên cửa sổ Command Window là:

All three formats produced the same content.

Ví dụ tiếp theo tạo ra mã vòng, cộng nhiễu và sau đó giải mã mã bị nhiễu. Nó sử dụng hàm giải mã.

```
n = 3, k = 2; % A (3,2) mã vòng
msg = randint(100,k,[0,1]); % 100 bản tin, k bit mỗi bản tin
code = encode(msg,n,k,'cyclic/binary');
% Cộng nhiễu.
noisycode = rem(code + randerr(100,n,[0 1; .7 .3]), 2);
newmsg = decode(noisycode,n,k,'cyclic'); % Cố gắng giải mã.
% Tính tốc độ lỗi để giải mã mã bị nhiễu
[number, ratio] = biterr(newmsg, msg);
disp(['The bit error rate is ', num2str(ratio)])
```

Đầu ra như dưới đây. Các kết quả tốc độ lỗi của bạn có thể khác do nhiễu là ngẫu nhiên.

```
The bit error rate is 0.08
```

Ví dụ tiếp theo mã hóa bản tin tương tự sử dụng các phương pháp Hamming, BCH, và mã vòng. Trước khi tạo ra mã BCH, nó sử dụng lệnh `bchpoly` để phát hiện ra các chiều dài bản tin và từ mã nào là hợp lệ. Ví dụ này cũng tạo ra mã Hamming với tùy chọn 'linear' của lệnh `encode`. Sau đó nó giải mã và khôi phục bản tin

```
n = 6; % Thử chiều dài từ mã = 6.
% Tìm bất kỳ chiều dài bản tin hợp lệ cho mã BCH.
params = bchpoly(n);
n = params(1,1); % Định nghĩa lại chiều dài từ mã trong trường hợp chiều dài từ %
mã trước đó không hợp lệ.
k = params(1,2); % Chiều dài bản tin
m = log2(n+1); % Biểu diễn n ở dạng 2^m-1
msg = randint(100,1,[0,2^k-1]); % Cột các số nguyên
% Tạo ra các mã khác nhau.
codehamming = encode(msg,n,k,'hamming/decimal'),
[parmat, genmat] = hamngen(m);
codehamming2 = encode(msg,n,k,'linear/decimal',genmat);
if codehamming==codehamming2
disp('The "linear" method can create Hamming code.')
```

```

end
codebch = encode(msg,n,k,'bch/decimal');
codecyclc = encode(msg,n,k,'cyclic/decimal');
% Giải mã để khôi phục lại bản tin ban đầu.
decodedhamming = decode(codehamming,n,k,'hamming/decimal');
decodedbch = decode(codebch,n,k,'bch/decimal');
decodedcyclic = decode(codecyclic,n,k,'cyclic/decimal');
if (decodedhamming==msg & decodedbch==msg & decodedcyclic==msg)
disp('All decoding worked flawlessly in this noiseless world.')
end

```

Kết quả trên cửa sổ Command Window:

The 'linear' method can create Hamming code.

All decoding worked flawlessly in this noiseless world.

Thuật toán phụ thuộc vào phương pháp mã hóa, encode phụ thuộc vào các hàm cấp thấp như hammgcn, cyclgen, and bchenco.

Xem thêm decode, bchpoly, cyclpoly, cyclgen, hammgcn, bchenco.

23.12 Tính khoảng cách cực tiểu của 1 mã khối tuyến tính

Khoảng cách cực tiểu, hay trọng lượng cực tiểu, của 1 mã khối tuyến tính được định nghĩa như là số dương nhỏ nhất của các phần tử khác không trong bất kỳ n-tuple mà là 1 từ mã.

$wt = gfweight(genmat)$ trả về khoảng cách tối thiểu của mã khối tuyến tính mà ma trận sinh của nó là genmat.

$wt = gfweight(genmat, 'gen')$ trả về khoảng cách cực tiểu của mã khối tuyến tính mà ma trận sinh là genmat.

$wt = gfweight(parmat, 'par')$ trả về khoảng cách cực tiểu của mã khối tuyến tính mà ma trận kiểm tra chẵn lẻ là parmat.

$wt = gfweight(genpoly, n)$ trả về khoảng cách cực tiểu mà đa thức sinh của nó được biểu diễn bởi genpoly. genpoly là vector hàng mà đưa ra các hệ số của đa thức sinh theo thứ tự bậc tăng.

Ví dụ

Các lệnh dưới đây minh họa 3 cách để tính khoảng cách cực tiểu của mã vòng (7,4).

$n = 7$:

```
% Đa thức sinh của mã vòng (7,4)
>> genpoly = cyclpoly(n,4),
>> [parmat, genmat] = cyclgen(n,genpoly);
>> wts = {gfweight(genmat,'gen'),gfweight(parmat,'par'),...
>> gfweight(genpoly,n)}
wts =
    3    3    3
```

23.13 Tạo ra ma trận sinh và ma trận kiểm tra chẵn lẻ cho mã Hamming

Hàm $h = \text{hammgen}(m)$ tạo ra ma trận kiểm tra chẵn lẻ $m \times n$ cho 1 mã Hamming có chiều dài từ mã $n = 2^m - 1$. Đầu vào m là số nguyên dương lớn hơn hoặc bằng 3. Chiều dài bản tin của mã là $n - m$. Đa thức nguyên tố nhị phân được sử dụng để sinh ra mã Hamming là đa thức nguyên tố mặc định cho $GF(2^m)$, được biểu diễn bởi $\text{gfprimdf}(m)$.

Hàm $h = \text{hammgen}(m, \text{pol})$ tạo ra một ma trận kiểm tra chẵn lẻ cho một mã Hamming có chiều dài từ mã $n = 2^m - 1$. Đầu vào m là số nguyên dương lớn hơn hoặc bằng 3. Chiều dài bản tin của mã là $n - m$. pol là một vector hàng mà đưa ra các hệ số theo thứ tự bậc tăng của đa thức gốc nhị phân cho $GF(2^m)$ mà được sử dụng để sinh ra mã Hamming

hammgen sinh ra một lỗi nếu pol biểu diễn 1 đa thức mà không phải là đa thức gốc

$[h, g] = \text{hammgen}(\dots)$ giống với $h = \text{hammgen}(\dots)$ ngoại trừ rằng nó cũng sinh ra ma trận sinh $k \times n$ là g mà tương ứng với ma trận kiểm tra chẵn lẻ h . k là chiều dài bản tin bằng

$n - m$, hoặc $2^m - 1 - m$.

$[h, g, n, k] = \text{hammgen}(\dots)$ giống với $[h, g] = \text{hammgen}(\dots)$ ngoại trừ rằng nó cũng trả về chiều dài từ mã n và chiều dài bản tin k .

Chú ý Nếu giá trị của m nhỏ hơn 25 thì đa thức gốc là đa thức gốc mặc định cho $GF(2^m)$ thì $\text{hammgen}(m)$ thực hiện nhanh hơn so với câu lệnh $\text{hammgen}(m, \text{pol})$.

Ví dụ Đoạn mã dưới đây đưa ra các ma trận sinh và kiểm tra chẵn lẻ cho 1 mã Hamming với chiều dài từ mã $7 = 2^3 - 1$ và chiều dài bản tin $4 = 7 - 3$.

```
[h,g,n,k] = hammgen(3)
h =
    1    0    0    1    0    1    1
    0    1    0    1    1    1    0
    0    0    1    0    1    1    1
```

g =

1 1 0 1 0 0 0

0 1 1 0 1 0 0

1 1 1 0 0 1 0

1 0 1 0 0 0 1

n =

7

k =

4

Đoạn mã dưới đây, mà sử dụng $1 + x^2 + x^3$ như là đa thức gốc cho GF(23), cho thấy rằng ma trận kiểm tra chẵn lẻ phụ thuộc vào sự lựa chọn đa thức nguyên tố. Chú ý rằng h1 dưới đây khác với h trong ví dụ trên.

h1 = hammgen(3,[1 0 1 1])

h1 =

1 0 0 1 1 1 0

0 1 0 0 1 1 1

0 0 1 1 1 0 1

23.14 Sinh ra bảng giải mã syndrome

t = syndtable(h) trả về 1 bảng giải mã cho mã nhị phân sửa lỗi có chiều dài từ mã n và chiều dài bản tin k. h là ma trận kiểm tra chẵn lẻ (n-k)xn. t là 1 ma trận nhị phân.

(2n-k)xn. Hàng thứ r của t là mẫu lỗi cho từ mã nhị phân nhận được mà syndrome có giá trị thập phân r-1. (syndrome của 1 từ mã nhận được là chuyển vị của ma trận kiểm tra chẵn lẻ) Nói cách khác, các hàng của t biểu diễn các tập coset từ dãy chuẩn của mã.

Khi chuyển đổi giữa các giá trị nhị phân và thập phân, cột ngoài cùng bên trái được hiểu như là số quan trọng nhất. Điều này khác với quy ước mặc định trong các lệnh bit2de và de2bit.

23.15 Đa thức sinh của mã Reed-Solomon

genpoly = rsgenpoly(n,k) trả về đa thức sinh của 1 mã Reed-Solomon với chiều dài từ mã n và chiều dài bản tin k. Chiều dài từ mã n phải có dạng $2^m - 1$ với m là số nguyên, và n-k phải là số nguyên chẵn. Đầu ra genpoly là 1 vector hàng Galois mà biểu diễn các hệ số của đa thức sinh theo thứ tự bậc giảm. Đa thức sinh narrow-sense generator là

$(X - A_1)(X - A_2) \dots (X - A_{2t})$ trong đó A là nghiệm của đa thức nguyên tố mặc định cho trường GF(n+1) và t là khả năng sửa lỗi. (n-k)/2

`genpoly = rsgenpoly(n,k,prim_poly)` giống với cú pháp trên, ngoại trừ rằng `prim_poly` chỉ ra đa thức nguyên tố cho $GF(n+1)$ mà có A là 1 nghiệm. `prim_poly` là số nguyên mà dạng biểu diễn nhị phân của nó chỉ ra các hệ số của đa thức nguyên tố. Để sử dụng đa thức nguyên tố mặc định $GF(n+1)$, thiết lập `prim_poly` là []

`genpoly = rsgenpoly(n,k,prim_poly,b)` trả về đa thức sinh

$(X - Ab)(X - Ab+1) \dots (X - Ab+2t-1)$ trong đó b là 1 số nguyên, A là nghiệm của `prim_poly` và t là khả năng sửa lỗi của mã, $(n-k)/2$.

`[genpoly,t] = rsgenpoly(...)` trả về t , là khả năng sửa lỗi của mã.

Ví dụ Các ví dụ dưới đây tạo ra các vector hàng Galois mà biểu diễn các đa thức sinh cho một mã Reed-Solomon [7,3]. Các vector g và $g2$ biểu diễn đa thức sinh narrow-sense, nhưng ứng với các phần tử nguyên tố khác nhau A . Cụ thể hơn, $g2$ được xác định như là A là nghiệm của đa thức nguyên tố $D^3 + D^2 + 1$ cho $GF(8)$. Vector $g3$ biểu diễn đa thức sinh $(X - A^3)(X - A^4)(X - A^5)(X - A^6)$, trong đó A là 1 nghiệm của $D^3 + D^2 + 1$ trong $GF(8)$.

```
>> g = rsgenpoly(7,3)
```

```
>> g = GF(2^3) array. Primitive polynomial = D^3+D+1 (11 decimal)
```

```
Array elements =
```

```
1 3 1 2 3
```

```
>> g2 = rsgenpoly(7,3,13) % Sử dụng đa thức gốc không mặc định.
```

```
>> g2 = GF(2^3) array. Primitive polynomial = D^3+D^2+1 (13 decimal)
```

```
Array elements =
```

```
1 4 5 1 5
```

```
g3 = rsgenpoly(7,3,13,3) % Sử dụng b = 3.
```

```
g3 = GF(2^3) array. Đa thức gốc = D^3+D^2+1 (13 decimal)
```

```
Array elements =
```

```
1 7 1 6 7
```

Các lệnh ví dụ sau sẽ thể hiện một khoảng cách mã gần:

Đa thức sinh cho mã Reed-Solomon [15,11] là $X^4 +$

$(A^3 + A^2 + 1)X^3 + (A^3 + A^2)X^2 + A^3X - (A^2 + A + 1)$ trong đó A là nghiệm của đa thức gốc với $GF(16)$.

```
gp = rsgenpoly(15,11)
```

```
gp = GF(2^4) array. Đa thức gốc = D^4+D+1 (19 decimal)
```

```
Array elements =
```

```
1 13 12 8 7
```

Giới hạn $n-k$ phải là số nguyên chẵn. Giá trị cho phép cực đại của n là 65535.

23.16 Giải mã chấp dữ liệu nhị phân sử dụng thuật toán Viterbi

Hàm `decoded = vitdec(code,trellis,tblen,opmode,dectype)` giải mã vector `code` sử dụng thuật toán Viterbi. Cấu trúc `trellis` trong MATLAB xác định bộ mã hóa chấp mà sinh ra mã; dạng của `trellis` được mô tả trong "Mô tả trellis của 1 bộ mã hóa chấp" và tài liệu tham khảo cho hàm `istrellis`. `code` có lớn hơn hoặc bằng 1 kí hiệu, mỗi kí hiệu có $\log_2(\text{trellis.numOutputSymbols})$ bit.

Mỗi kí hiệu trong vector `decoded` có $\log_2(\text{trellis.numInputSymbols})$ bit. `tblen` là số nguyên dương mà chỉ ra độ sâu traceback.

Xâu `opmode` chỉ ra chế độ hoạt động của bộ giải mã và giả thiết của nó về hoạt động của bộ mã hóa tương ứng. Các lựa chọn có trong bảng dưới đây.

Giá trị	Ý nghĩa
'Trunc'	Bộ mã hóa được thiết lập tất cả các trạng thái ban đầu là 0. Bộ giải mã quét ngược từ trạng thái với ma trận phù hợp nhất.
'term'	Bộ mã hóa được giả thiết là đã khởi động và kết thúc ở trạng thái 0. Bộ giải mã quét ngược từ trạng thái toàn bộ 0.
'cont'	Bộ mã hóa được giả thiết là đã bắt đầu ở trạng thái toàn bộ là 0. Bộ giải mã quét ngược từ trạng thái với ma trận phù hợp nhất Một trễ bằng <code>tblen</code> kí hiệu xảy ra trước khi kí hiệu được giải mã đầu tiên xuất hiện ở đầu ra.

Xâu `dectype` chỉ ra loại quyết định mà bộ giải mã thực hiện, và loại dữ liệu mà bộ giải mã nhận được trong `code`. Các lựa chọn như trong bảng dưới đây.

Value	Meaning
'unquant'	code có các giá trị đầu vào thực, trong đó 1 biểu diễn giá trị logic 0 và -1 biểu diễn giá trị logic 1.
'hard'	code có các giá trị đầu vào nhị phân.
'soft'	Để giải mã quyết định mềm sử dụng cú pháp dưới đây. Chú ý rằng <code>nsdec</code> được yêu cầu cho giải mã quyết định mềm.

Cú pháp cho giải mã quyết định mềm

`decoded = vitdec(code,trellis,tblen,opmode 'soft',nsdec)` giải mã vector `code` sử dụng giải mã quyết định mềm. `code` gồm các số nguyên trong khoảng 0 và $2^{nsdec}-1$, trong đó 0 biểu diễn giá trị tin cậy nhất cho 0 và $2^{nsdec}-1$ biểu diễn giá trị tin cậy nhất cho 1.

Các cú pháp khác cho chế độ hoạt động liên tục

`decoded = vitdec(...,'cont', ...,initmetric,initstates,initinputs)` giống với các cú pháp trước đó, ngoại trừ rằng bộ giải mã bắt đầu với các ma trận trạng thái của nó, các trạng thái traceback, và các đầu vào traceback được chỉ ra trong `initmetric`, `initstates`, và `initinputs`

Mỗi số thực trong `initmetric` biểu diễn ma trận trạng thái bắt đầu của trạng thái tương ứng. `initstates` và `initinputs` cung chỉ ra nhớ traceback ban đầu của bộ giải mã; cả hai là các ma trận `trellis.numStates-by-tblen`. `initstates` gồm các số nguyên trong khoảng 0 và `trellis.numStates-1`. Nếu mạch điện bộ mã hóa có nhiều hơn 1 dòng đầu vào thì thanh ghi dịch mã nhận đường vào đầu tiên đưa ra các bit LSB trong `initstates`, trong khi thanh ghi dịch mã nhận đường vào cuối cùng đưa ra các bit MSB trong `initstates`. Vector `initinputs` gồm các số nguyên trong khoảng 0 và `trellis.numInputSymbols-1`. Để sử dụng các giá trị mặc định cho tất cả 3 đối số cuối cùng thiết lập chúng là `[],[],[]`.

`[decoded,finalmetric,finalstates,finalinputs] = ...vitdec(...,'cont',...)` giống với các cú pháp trên, ngoại trừ rằng 3 đối số đầu ra cuối cùng trả về các ma trận trạng thái, các trạng thái traceback, và các đầu vào traceback, ở cuối của quá trình giải mã. `finalmetric` là 1 vector với các phần tử `trellis.numStates` mà tương ứng với các ma trận trạng thái cuối cùng. `finalstates` và `finalinputs` là các ma trận kích thước `trellis.numStates-by-tblen`. Các phần tử của `finalstates` có định dạng giống như trong `initstates`.

Ví dụ Ví dụ dưới đây mã hóa dữ liệu ngẫu nhiên và cộng nhiễu. Sau đó nó giải mã mã bị nhiễu 3 lần và minh họa 2 loại quyết định mà `vitdec` hỗ trợ. Chú ý rằng cho các quyết định mềm và không lượng tử hóa, đầu ra của `convenc` không cùng loại dữ liệu mà `vitdec` nhận được trong mã vào.

```
>> trell = poly2trellis(3,[6 7]); % Định nghĩa trellis.  
>> msg = randint(100,1,2,123); % Dữ liệu ngẫu nhiên  
>> code = convenc(msg,trell); % Mã hóa.  
>> ncode = rem(code + randerr(200,1,[0 1; .95 .05]),2); % Cộng nhiễu.  
>> tblen = 3; % Chiều dài Traceback
```

Sử dụng g ãi mã quyết định cứng

```
>> decoded1 = vitdec(ncode,trell,tblen,'cont','hard');
```

Sử dụng giải mã quyết định chưa được lượng tử hóa.

```
>> ucode = 1-2*ncode; % +1 & -1 represent zero & one, respectively.
```

```
>> decoded2 = vitdec(ucode,trell,tblen,'cont','unquant');
```

Sử dụng giải mã quyết định mềm.

Để chuẩn bị cho giải mã quyết định mềm, ánh xạ sang các giá trị quyết định.

```
>> [x,qcode] = quantiz(1-2*ncode,[-.75 -5 25 0 25 .5 .75],...)
```

[7 6 5 4 3 2 1 0]); % Các giá trị trong qcode nằm trong khoảng 0 và 2^3-1 .

```
>> decoded3 = vitdec(qcode',trell,tblen,'cont','soft',3);
```

Tính các tốc độ lỗi bit khi đầu ra bộ giải mã bị trễ đi tblen symbols.

```
>> [n1,r1] = biterr(decoded1(tblen+1:end),msg(1:end-tblen));
```

```
>> [n2,r2] = biterr(decoded2(tblen+1:end),msg(1:end-tblen));
```

```
>> [n3,r3] = biterr(decoded3(tblen+1:end),msg(1:end-tblen));
```

```
>> disp(['The bit error rates are: ',num2str([r1 r2 r3])])
```

Kết quả:

The bit error rates are: 0.020619 0.020619 0.020619

Ví dụ dưới đây minh họa làm thế nào để sử dụng các đối số trạng thái cuối cùng và trạng thái ban đầu khi thực hiện vitdec 1 cách lặp lại. Chú ý rằng [decoded4,decoded5] giống với decoded6.

```
>> trell = poly2trellis(3,[6 7]);
```

```
>> code = convenc(randint(100,1,2,123),trell);
```

Giải mã một phần của mã code, theo trạng thái cuối cùng để sử dụng sau này.

```
>> [decoded4,f1,f2,f3] = vitdec(code(1:100),trell,3,'cont','hard');
```

Giải mã phần còn lại của mã code, sử dụng các đối số đầu vào trạng thái.

```
>> decoded5 = vitdec(code(101:200),trell,3,'cont','hard',f1 f2,f3);
```

Giải mã toàn bộ mã trong 1 bước.

```
>> decoded6 = vitdec(code,trell,3,'cont','hard');
```

```
>> isequal(decoded6,[decoded4; decoded5])
```

ans =

1

23.17 Điều chế băng thông tương tự

Các cú pháp lệnh:

Hàm $y = \text{amod}(x,F_c,F_s,\text{'amdsb-tc'},\text{offset})$ thực hiện điều biên. offset là giá trị cộng thêm vào x trước khi điều chế. Nếu bỏ qua offset, thì giá trị mặc định của nó là $-\min(\min(x))$. Giá trị mặc định này tạo ra điều chế 100%.

$y = \text{amod}(x,F_c,F_s,\text{'amdsb-sc'})$ thực hiện điều chế biên độ sóng mang nên dài gấp.

$y = \text{amod}(x, F_c, F_s, 'amssb/opt')$ thực hiện điều chế biên độ sóng mang nên đơn biên. Mặc định nó tạo ra dải dưới, nếu *opt* là **up**, thì hàm sinh ra dải tần trên (biên trên). Cú pháp này không thực hiện chuyển đổi Hilbert trong miền tần số.

$y = \text{amod}(x, F_c, F_s, 'amssb/opt', \text{num}, \text{den})$ giống với cú pháp ở trên, ngoại trừ nó chỉ ra bộ lọc Hilbert miền thời gian. *num* và *den* là các vector hàng mà chứa các hệ số, theo thứ tự bậc giảm dần, của tử số và mẫu số của hàm truyền bộ lọc. Có thể sử dụng hàm `hilbirt` để thiết kế bộ lọc Hilbert.

$y = \text{amod}(x, F_c, F_s, 'amssb/opt', \text{hilbertflag})$ giống với cú pháp trên, ngoại trừ nó sử dụng mặc định bộ lọc miền thời gian Hilbert. Hàm truyền của bộ lọc được xác định bởi $[\text{num}, \text{den}] = \text{hilbirt}(1/F_s)$, trong đó *num* và *den* được nói đến ở trên. Đối số vào *hilbertflag* có thể nhận giá trị bất kỳ

$y = \text{amod}(x, F_c, F_s, 'qam')$ thực hiện điều chế biên độ cầu phương. *x* là 1 ma trận 2 cột mà cột thứ nhất biểu diễn tín hiệu cùng pha và cột thứ 2 biểu diễn tín hiệu pha vuông góc. *y* là vector cột.

$y = \text{amod}(x, F_c, F_s, 'fm', \text{deviation})$ thực hiện điều chế tần số. Phổ của tín hiệu điều chế nằm trong khoảng $\min(x) + F_c$ và $\max(x) + F_c$.

Đối số tùy chọn *deviation* là 1 số vô hướng mà biểu diễn hằng số độ lệch tần số của điều chế. Lệnh $y = \text{amod}(x, F_c, F_s, 'fm', \text{deviation})$ tương đương với lệnh.

$y = \text{amod}(x * \text{deviation}, F_c, F_s, 'fm')$.

$y = \text{amod}(x, F_c, F_s, 'pm', \text{deviation})$ thực hiện điều chế pha. Đối số tùy chọn *deviation* là số vô hướng biểu diễn hằng số độ lệch pha của điều chế.

Lệnh $y = \text{amod}(x, F_c, F_s, 'pm', \text{deviation})$ tương đương với lệnh

$y = \text{amod}(x * \text{deviation}, F_c, F_s, 'pm')$.

$[y, t] = \text{amod}(\dots)$ trả về thời gian tính trong biến *t*.

Ví dụ so sánh điều chế dải đơn và dải kép

Ví dụ đầu tiên so sánh phổ của các tín hiệu sau điều chế sử dụng kỹ thuật điều chế dải đơn và dải kép. Tín hiệu bản tin là 1 sóng sin 1 tần số và tín hiệu sóng mang là sóng sin 10 Hz. Đoạn lệnh dưới đây sử dụng đối số **'amdsb-sc'** và **'amssb'** trong hàm `amod` để sinh ra các tín hiệu điều chế *ydouble* và *ysingle*. Sau đó nó vẽ phổ của cả 2 tín hiệu điều chế.

% Lấy mẫu tín hiệu 100 lần/s, trong 2 giây.

Fs = 100;

t = [0:2*Fs+1]/Fs;

Fc = 10; % Tần số sóng mang

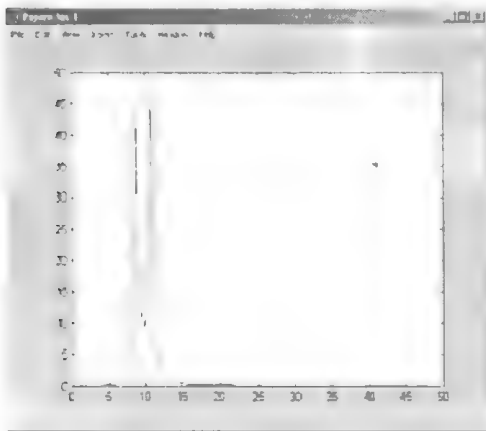
```

x = sin(2*pi*t);
% Tín hiệu sin
% Điều chế x sử dụng điều chế biên độ dải đơn và dải kép
ydouble = amod(x,Fc,Fs,'amdsb-sc');
ysingle = amod(x,Fc,Fs,'amssb');
% Vẽ phổ của cả 2 tín hiệu điều chế
zdouble = fft(ydouble);
zdouble = abs(zdouble(1:length(zdouble)/2+1));
frqdouble = [0 length(zdouble)-1]*Fs/length(zdouble)/2;
plot(frqdouble,zdouble);
% Hình bên trái trong hình vẽ dưới đây
zsingle = fft(ysingle);
zsingle = abs(zsingle(1:length(zsingle)/2+1));
frqsingle = [0:length(zsingle)-1]*Fs/length(zsingle)/2;
plot(frqsingle,zsingle);

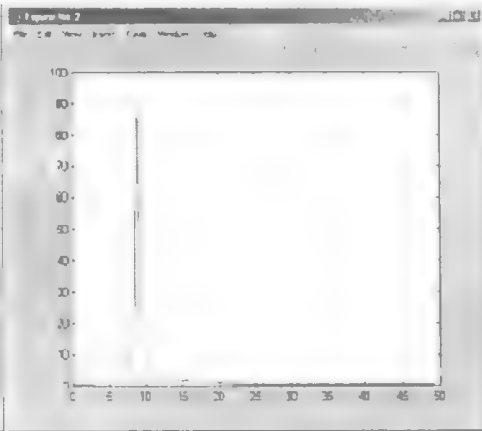
```

% Hình vẽ bên phải trong hình vẽ dưới đây.

Chú ý rằng phổ ở hình bên trái có 2 đỉnh, đây là các biên tần của tín hiệu điều chế, bao gồm biên tần trên và biên tần dưới. Những băng tần này đối xứng nhau qua tần số sóng mang F_c là 10Hz. Phổ của 1 tín hiệu điều chế DSB-SC AM rộng gấp 2 lần băng thông của tín hiệu vào. Trong hình vẽ thứ 2, chỉ có 1 đỉnh bởi vì kỹ thuật SSB AM chỉ cần truyền đi 1 băng.



Hình 23.3a



Hình 23.3b

23.18 Giải điều chế bằng thông tương tự

$z = \text{admod}(y, F_c, F_s, 'amdsb-tc', \text{offset}, \text{num}, \text{den})$ thực hiện giải điều chế biên độ dải kép, offset là 1 vector mà thành phần thứ k của nó được trừ từ tín hiệu thứ k sau khi điều chế. Nếu trường offset trống thì z mặc định được điều chỉnh để mỗi cột có trị trung bình bằng 0

$z = \text{admod}(y, F_c, F_s, 'amdsb-tc/costas', \text{offset}, \text{num}, \text{den})$ giống ở trên, ngoại trừ thuật toán có vòng khóa pha Costas.

$z = \text{admod}(y, F_c, F_s, 'amdsb-sc', \text{num}, \text{den})$ thực hiện giải điều chế biên độ sóng mang nén dải kép.

$z = \text{admod}(y, F_c, F_s, 'amdsb-sc/costas', \text{num}, \text{den})$ giống cú pháp ở trên, ngoại trừ thuật toán bao gồm vòng khóa pha Costas.

$z = \text{admod}(y, F_c, F_s, 'amssb', \text{num}, \text{den})$ thực hiện giải điều chế biên độ sóng mang nén đơn dải.

$z = \text{admod}(y, F_c, F_s, 'qam', \text{num}, \text{den})$ thực hiện giải điều chế biến độ cầu phương.

$z = \text{admod}(y, F_c, F_s, 'fm', \text{num}, \text{den}, \text{vcoconst})$ thực hiện giải điều chế tần số. Phổ của tín hiệu giải điều chế nằm trong khoảng $\min(y) + F_c$ và $\max(y) + F_c$. Quá trình giải điều chế sử dụng vòng khóa pha gồm 1 bộ nhân (đóng vai trò bộ phát hiện pha), 1 bộ lọc thông thấp, và 1 bộ dao động điều chỉnh bằng điện áp (VCO). Nếu F_s là vector 2 phần tử thì phần tử thứ 2 là pha ban đầu của VCO tính theo radians. Đối số vcoconst là số vô hướng mà biểu diễn hằng số VCO tính theo Hz/V.

$z = \text{admod}(y, F_c, F_s, 'pm', \text{num}, \text{den}, \text{vcoconst})$ thực hiện giải điều chế pha. Quá trình giải điều chế sử dụng vòng khóa pha (mà hoạt động như trong bộ giải điều chế FM). Đối số tùy chọn vcoconst là số vô hướng biểu diễn độ nhạy tín hiệu vào.

Ví dụ Ví dụ này minh họa cách sử dụng đối số offset . Bởi vì dòng lệnh đầu tiên sử dụng cùng giá trị offset là .3 mà lệnh amod đã sử dụng, $z1$ giống với tín hiệu bản tin ban đầu. Bởi vì lệnh thứ 2 admod bỏ qua offset , $z2$ có trị trung bình gần bằng 0 (không chính xác bằng 0 vì lỗi làm tròn).

$F_c = 25$; % Tần số tín hiệu sóng mang

$F_s = 100$, % Tần số lấy mẫu của tín hiệu

$t = [0:1/F_s:5]'$; % Thời gian lấy mẫu tín hiệu

$x = [\cos(t), \sin(t)]$; % Tín hiệu Cosine và sine

$y = \text{amod}(x, F_c, F_s, 'amdsb-tc', .3)$; % Điều chế

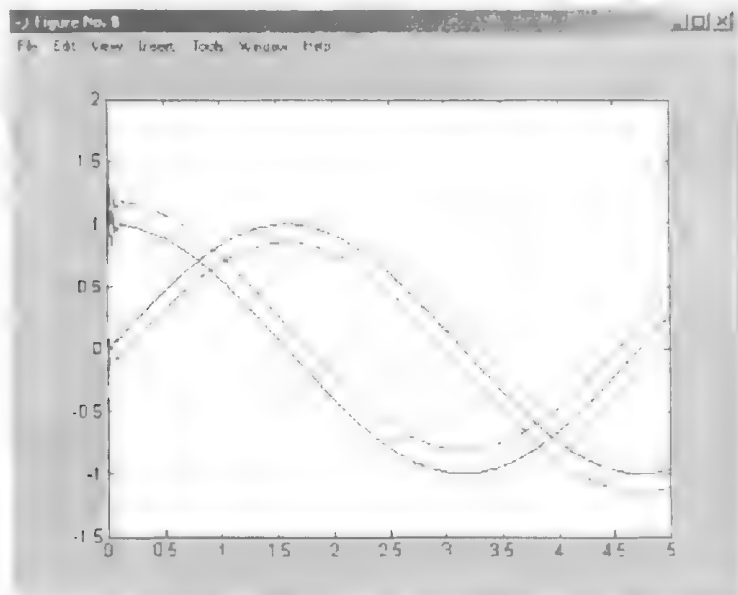
% and shift the values up by .3

$z1 = \text{admod}(y, F_c, F_s, 'amdsb-tc', .3)$; % Giải điều chế

$z2 = \text{admod}(y, F_c, F_s, 'amdsb-tc')$; % Giải điều chế

`plot(t,z1,'b',t,z2,'r-') % Vẽ tín hiệu được khôi phục`

Hình vẽ hiển thị z1 là đường vẽ liền nét và z2 là đường vẽ gạch nét.



Hình 23.4

23.19 Bộ điều chế băng thông số

Điều chế 1 tín hiệu số (Thông tin về cú pháp cụ thể).

$y = \text{dmod}(x, F_c, F_d, F_s, \text{'ask'}, M)$ thực hiện điều chế khóa dịch biên độ M mức. Mỗi phần tử của x phải nằm trong khoảng $[0, M-1]$. Giá trị cực đại của tín hiệu được điều chế là 1.

$y = \text{dmod}(x, F_c, F_d, F_s, \text{'fsk'}, M, \text{tone})$ thực hiện điều chế khóa dịch tần số M mức. Mỗi phần tử của x nằm trong khoảng $[0, M-1]$. Đối số tùy chọn tone là phân cách giữa các tần số kế tiếp trong tín hiệu điều chế y . Giá trị mặc định của tone là F_d . Giá trị cực đại của y là 1.

$y = \text{dmod}(x, F_c, F_d, F_s, \text{'msk'})$ thực hiện điều chế khóa dịch cực tiểu. Mỗi phần tử của x là 0 hoặc 1. Giá trị cực đại của y là 1.

$y = \text{dmod}(x, F_c, F_d, F_s, \text{'psk'}, M)$ thực hiện điều chế dịch pha M mức. Mỗi phần tử của x phải nằm trong khoảng $[0, M-1]$. Giá trị cực đại của y là 1.

$y = \text{dmod}(x, F_c, F_d, F_s, \text{'qask'}, M)$ thực hiện điều chế khóa dịch biên độ cầu phương M mức với đồ thị chòm sao tín hiệu vuông. Bảng dưới đây cho thấy các giá trị cực đại của y ứng với một số giá trị của M .

M	Giá trị cực đại của y	M	Giá trị cực đại của y
2	1	32	5
4	1	64	7
8	3	128	11
16	3	256	15

Chú ý

Để thấy có bao nhiêu kí hiệu được biến đổi vào các điểm chòm sao, sinh ra 1 đồ thị chòm sao vuông sử dụng `qaskenco(M)`.

`y = dmod(x,Fc,Fd,Fs,'qask/arb',inphase,quadr)` thực hiện điều chế khóa dịch biên độ cầu phương với đồ thị chòm sao được định nghĩa sử dụng các vector `inphase` và `quadr`. Điểm chòm sao cho bản tin thứ `k` có thành phần cùng pha `inphase(k+1)` và thành phần cầu phương `quadr(k+1)`.

`y = dmod(x,Fc,Fd,Fs,'qask/cir',numsig,amp,phs)` thực hiện điều chế khóa dịch biên độ cầu phương với 1 đồ thị chòm sao tín hiệu tròn. `numsig`, `amp`, và `phs` là các vector có cùng chiều dài. Các phần tử trong `numsig` và `amp` phải dương. Nếu `k` là 1 số nguyên trong khoảng `[1, length(numsig)]`, thì `amp(k)` là bán kính của vòng tròn thứ `k`, và `phs(k)` là pha của điểm chòm sao đầu tiên trên vòng tròn thứ `k`. Tất cả các điểm trên vòng tròn thứ `k` được xếp cách đều nhau. Nếu bỏ qua `phs`, thì giá trị mặc định của nó là `numsig*0`. Nếu bỏ qua `amp`, thì giá trị mặc định của nó là `[1:length(numsig)]`.

Chú ý Để thấy có bao nhiêu kí hiệu được biến đổi thành các điểm chòm sao, sinh ra 1 đồ thị chòm sao tròn sử dụng `apkconst(numsig,amp,phs,'n')`.

`[y,t] = dmod(...)` trả về thời gian tính toán trong `t`. `t` là 1 vector mà chiều dài của nó là số hàng của `y`.

Ví dụ Một ví dụ trong phần tham chiếu cho `ddemod` sử dụng `dmod`. Cũng như vậy đoạn mã dưới đây hiển thị dạng sóng được sử dụng để truyền các số 0 và 1 sử dụng điều chế

4-ASK. Chú ý rằng lệnh `dmod` có 2 đối số đầu ra. Đầu ra thứ 2, `t` được sử dụng để định cỡ trục ngang trong hình vẽ.

`Fc = 20; Fd = 10; Fs = 50;`

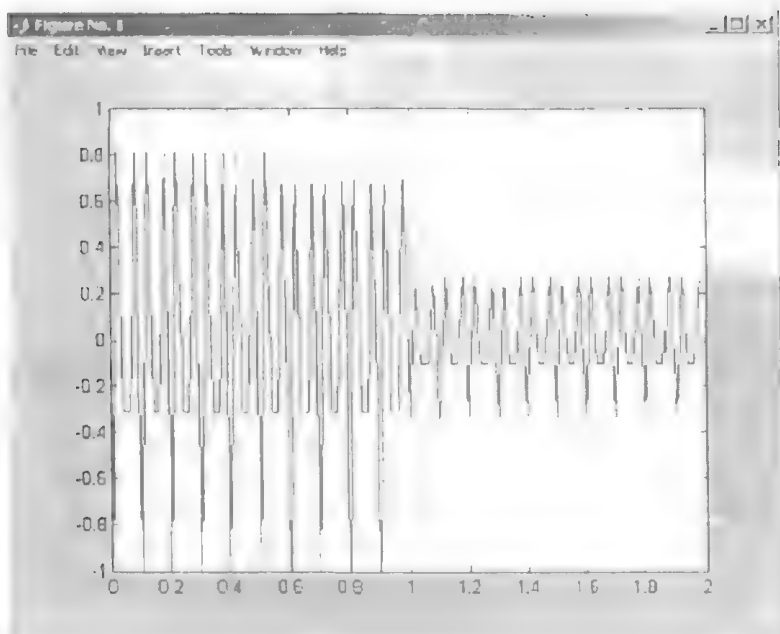
`M = 4, % Sử dụng điều chế 4-ASK.`

`x = ones(Fd,1)*[0 1]; x=x(:);`

`% Điều chế, trong miền thời gian.`

`[y,t] = dmod(x,Fc,Fd,Fs,'ask',M);`

`plot(t,y) % Vẽ tín hiệu theo thời gian.`



Hình 23.5

23.20 Bộ điều chế băng gốc số.

$y = \text{dmodce}(x, F_d, F_s, 'ask', M)$ thực hiện điều chế khóa dịch biên độ M mức. Mỗi phần tử của x nằm trong khoảng $[0, M-1]$. Giá trị cực đại của tín hiệu điều chế là 1.

$y = \text{dmodce}(x, F_d, F_s, 'fsk', M, \text{tone})$ thực hiện điều chế khóa dịch tần số M mức. Mỗi phần tử của x nằm trong khoảng $[0, M-1]$. Đối số tùy chọn tone là phân cách giữa các tần số kế tiếp trong tín hiệu điều chế y . Giá trị mặc định của tone là F_d . Giá trị cực đại của y là 1.

$y = \text{dmodce}(x, F_d, F_s, 'msk')$ thực hiện điều chế khóa dịch cực tiểu. Mỗi phần tử của x là 0 hoặc 1. Phân cách giữa 2 tần số là $F_d/2$.

$y = \text{dmodce}(x, F_d, F_s, 'psk', M)$, thực hiện điều chế khóa dịch pha M mức. Mỗi phần tử của x nằm trong khoảng $[0, M-1]$. Giá trị cực đại của y là 1.

$y = \text{dmodce}(x, F_d, F_s, 'qask', M)$ thực hiện điều chế khóa dịch biên độ cầu phương M mức với đồ thị chòm sao tín hiệu vuông. Bảng dưới đây liệt kê cả giá trị cực đại của y ứng với một số giá trị của M .

M	Giá trị cực đại của y	M	Giá trị cực đại của y
2	1	32	5
4	1	64	7

8	3	128	11
16	3	256	15

Chú ý Để thấy có bao nhiêu kí hiệu được biến đổi thành các điểm chòm sao, sinh ra 1 đồ thị chòm sao vuông sử dụng qaskenco(M).

$y = \text{dmodce}(x, F_d, F_s, 'qask/arb', \text{inphase}, \text{quadr})$ thực hiện điều chế khóa dịch biên độ cầu phương, với 1 đồ thị chòm sao tín hiệu mà xác định trong các vector inphase và quadr. Điểm chòm sao cho bản tin thứ k có thành phần cùng pha inphase(k+1) và thành phần pha vuông góc quadr(k+1).

$y = \text{dmodce}(x, F_d, F_s, 'qask/cir', \text{numsig}, \text{amp}, \text{phs})$ thực hiện điều chế khóa dịch biên độ cầu phương với đồ thị chòm sao tín hiệu tròn. numsig, amp, và phs là các vector có cùng chiều dài. Các phần tử trong numsig và amp phải dương. Nếu k là 1 số nguyên trong khoảng [1, length(numsig)], thì amp(k) là bán kính của vòng tròn thứ k, và phs(k) là pha của điểm chòm sao đầu tiên trên vòng tròn thứ k. Tất cả các điểm trên vòng tròn thứ k được xếp cách đều nhau. Nếu bỏ qua phs thì giá trị mặc định của nó là numsig*0. Nếu bỏ qua amp, thì giá trị mặc định của nó là [1:length(numsig)].

Ví dụ Ví dụ này sử dụng điều chế và giải điều chế FSK với các giá trị khác của phân cách tần số, tone. Các kết quả của bạn có thể khác do ví dụ sử dụng các số ngẫu nhiên.

```
M = 4; Fd = 1; Fs = 32;
```

```
SNRperBit = 5;
```

```
adjSNR = SNRperBit-10*log10(Fs/Fd)+10*log10(log2(M));
```

```
x = randint(5000,1,M); % Tín hiệu ban đầu
```

```
% Điều chế sử dụng FSK với tần số sóng mang trực giao.
```

```
tone = .5;
```

```
randn('state',1945724); % Khởi tạo khối phát Gaussian.
```

```
w1 = dmodce(x,Fd,Fs,'fsk',M,tone);
```

```
y1 = awgn(w1, adjSNR, 'measured', [], 'dB');
```

```
z1 = ddemodce(y1,Fd,Fs,'fsk',M,tone);
```

```
ser1 = symerr(x,z1)
```

```
% điều chế sử dụng FSK với các tần số sóng mang không trực giao.
```

```
tone = .25;
```

```
randn('state',1945724); % Khởi tạo lại khối phát Gaussian.
```

```
w2 = dmodce(x,Fd,Fs,'fsk',M,tone);
```

```
y2 = awgn(w2, adjSNR, 'measured', [], 'dB');
```

```
z2 = ddemodce(y2,Fd,Fs,'fsk',M,tone),
```

```
ser2 = symerr(x,z2)
```

Kết quả nhận được

```
ser1 =
```

```
67
```

```
ser2 =
```

```
258
```

23.21 Bộ giải điều chế băng thông số

Để giải điều chế 1 tín hiệu số (Thông tin cú pháp cụ thể):

$z = \text{ddemod}(y, F_c, F_d, F_s, 'ask', M)$ thực hiện giải điều chế khóa dịch biên độ M mức. Mỗi phần tử của z nằm trong khoảng $[0, M-1]$.

$z = \text{ddemod}(y, F_c, F_d, F_s, 'ask/costas', M)$ giống với cú pháp ở trên, ngoại trừ rằng thuật toán bao gồm 1 vòng lặp Costas.

$z = \text{ddemod}(y, F_c, F_d, F_s, 'fsk', M, \text{tone})$ thực hiện giải điều chế khóa dịch tần số M mức. Đối số tùy chọn tone là khoảng phân cách giữa các tần số kế tiếp trong tín hiệu điều chế z . Giá trị mặc định của tone là F_d . Mỗi phần tử của z nằm trong khoảng $[0, M-1]$.

$z = \text{ddemod}(y, F_c, F_d, F_s, 'fsk/noncoherence', M, \text{tone})$ giống với cú pháp ở trên, ngoại trừ rằng nó sử dụng giải điều chế không liên kết.

$z = \text{ddemod}(y, F_c, F_d, F_s, 'msk')$ thực hiện giải điều chế dịch tối thiểu. Mỗi phần tử của z là 0 hoặc 1. Khoảng phân cách giữa 2 tần số là $F_d/2$.

$z = \text{ddemod}(y, F_c, F_d, F_s, 'psk', M)$ thực hiện giải điều chế khóa dịch pha tương quan M mức. Mỗi phần tử của z trong khoảng $[0, M-1]$.

$z = \text{ddemod}(y, F_c, F_d, F_s, 'qask', M)$ thực hiện giải điều chế khóa dịch biên độ M mức với đồ thị chòm sao vuông. Bảng dưới đây ddemod có thể sử dụng 1 bộ lọc thông thấp với chu kỳ lấy mẫu $1/F_s$ trong khi giải điều chế, để lọc bỏ tín hiệu sóng mang. Để chỉ ra bộ lọc thông thấp, sử dụng đối số đầu vào num và den là các vector hàng mà đưa ra các hệ số theo thứ tự bậc giảm của tử số và mẫu số của hàm truyền bộ lọc.

Để giải điều chế 1 tín hiệu số.

$z = \text{ddemod}(y, F_c, F_d, F_s, 'ask', M)$ thực hiện giải điều chế khóa dịch biên độ M mức. Mỗi giá trị của z nằm trong khoảng $[0, M-1]$.

$z = \text{ddemod}(y, F_c, F_d, F_s, 'ask/costas', M)$ có cú pháp giống như trên, ngoại trừ thuật toán bao gồm vòng lặp Costas.

$z = \text{ddemod}(y, F_c, F_d, F_s, \text{'fsk'}, M, \text{tone})$ thực hiện giải điều chế khóa dịch tần số M mức. Đối số tùy chọn tone là khoảng phân cách giữa các tần số liên tiếp trong tín hiệu điều chế z . Giá trị mặc định của tone là F_d . Mỗi phần tử của z nằm trong khoảng $[0, M-1]$.

$z = \text{ddemod}(y, F_c, F_d, F_s, \text{'fsk/noncoherence'}, M, \text{tone})$ có cú pháp giống trên, ngoại trừ rằng nó sử dụng giải điều chế không liên kết.

$z = \text{ddemod}(y, F_c, F_d, F_s, \text{'msk'})$ thực hiện giải điều chế khóa dịch cực tiểu. Mỗi phần tử của z là 0 hoặc 1. Khoảng cách phân cách giữa 2 tần số là $F_d/2$.

$z = \text{ddemod}(y, F_c, F_d, F_s, \text{'psk'}, M)$ thực hiện giải điều chế khóa dịch pha tương quan M mức. Mỗi phần tử của z nằm trong khoảng $[0, M-1]$.

$z = \text{ddemod}(y, F_c, F_d, F_s, \text{'qask'}, M)$ thực hiện giải điều chế khóa dịch biên độ cầu phương với 1 đồ thị chòm sao tín hiệu vuông.

M	Maximum of Coordinates of Constellation Points	M	Maximum of Coordinates of Constellation Points
2	1	32	5
4	1	64	7
8	3	128	11
16	3	256	15

Chú ý

Để xem có bao nhiêu kí hiệu được ánh xạ vào trong chòm sao, sinh ra 1 đồ thị chòm sao sử dụng `qaskenco(M)`.

$z = \text{ddemod}(y, F_c, F_d, F_s, \text{'qask/arb'}, \text{inphase}, \text{quadr})$ thực hiện giải điều chế khóa dịch biên độ cầu phương, với 1 đồ thị chòm sao tín hiệu mà định nghĩa sử dụng các vector `inphase` và `quadr`. Điểm trên đồ thị chòm sao cho bản tin thứ k có thành phần cùng pha `inphase(k+1)` và thành phần pha vuông góc `quadr(k+1)`.

$z = \text{ddemod}(y, F_c, F_d, F_s, \text{'qask/cir'}, \text{numsig}, \text{amp}, \text{phs})$ thực hiện giải điều chế khóa dịch biên độ cầu phương với 1 đồ thị chòm sao tín hiệu tròn. `numsig`, `amp`, và `phs` là các vector có cùng chiều dài. Các phần tử trong `numsig` và `amp` phải dương. Nếu k là một số nguyên trong khoảng $[1, \text{length}(\text{numsig})]$, thì `amp(k)` là bán kính của vòng tròn thứ k , `numsig(k)` là số điểm trên đồ thị chòm sao trên vòng tròn thứ k , và `phs(k)` là pha của điểm đầu tiên được vẽ trên vòng tròn thứ k . Tất cả các điểm trên vòng tròn thứ k là chẵn. Nếu bỏ qua `phs`, thì giá trị mặc định của nó là `numsig*0`. Nếu bỏ qua thì giá trị mặc định của nó là `[1:length(numsig)]`.

Chú ý Để thấy có bao nhiêu kí hiệu được ánh xạ tới các điểm trên đồ thị chòm sao, sinh ra 1 đồ thị chòm sao sử dụng `apkconst(numsig,amp,phs,'n')`.

Ví dụ Ví dụ này gần giống với ví dụ trong "Ví dụ điều chế số đơn giản" nhưng sử dụng mô phỏng băng thông. Nó sinh ra 1 tín hiệu số ngẫu nhiên, điều chế nó sử dụng `dmmod`, và cộng nhiễu. Sau đó nó giải điều chế tín hiệu mang nhiễu và tính tốc độ lỗi kí

hiệu. Hàm `ddemod` giải điều chế tín hiệu tương tự y và biến đổi ngược để sinh ra tín hiệu số z .

Các điểm khác nhau quan trọng giữa ví dụ này và ví dụ bằng gốc ban đầu là tham chiếu thẳng tới tần số tín hiệu sóng mang F_c và sự thực rằng y và y_{noisy} là số thực, không phải phức. Ví dụ này sử dụng ASK thay cho PSK, cũng như tốc độ lấy mẫu khác F_d .

```
M = 16; % Sử dụng điều chế 16 mức.
```

```
Fc = 10; % Tần số tín hiệu sóng mang là 10 Hz.
```

```
Fd = 1; % Tốc độ lấy mẫu của tín hiệu ban đầu và tín hiệu được điều chế
```

```
Fs = 50; % thứ tự là 1 và 50 (mẫu/s).
```

```
x = randint(100,1,M); % Bản tin số ngẫu nhiên
```

```
% Sử dụng điều chế PSK M mức để sinh ra y.
```

```
y = dmod(x,Fc,Fd,Fs,'ask',M);
```

```
% Cộng nhiễu Gaussian.
```

```
y_noisy = y + .01*randn(Fs/Fd*100,1);
```

```
% Giải điều chế để khôi phục bản tin.
```

```
z = ddemod(y_noisy,Fc,Fd,Fs,'ask',M);
```

```
s = symerr(x,z) % Kiểm tra tốc độ lỗi kí hiệu.
```

```
s =
```

```
0
```

23.22 Bộ giải điều chế băng gốc số

Mã hóa và giải mã DPCM. Một trường hợp riêng đơn giản của lượng tử hóa DPCM là sự khác nhau giữa giá trị tín hiệu hiện tại và giá trị của nó ở bước trước đó. Do đó bộ dự đoán chỉ là $y(k) = x(k-1)$. Đoạn mã bên dưới cài đặt sơ đồ này. Nó mã hóa một tín hiệu răng cưa, giải mã nó và vẽ đồng thời tín hiệu gốc và tín hiệu đã mã hóa. Đường đặc là tín hiệu gốc còn đường gạch là tín hiệu đã khôi phục. Ví dụ cũng tính sai số trung bình bình phương giữa các tín hiệu gốc và đã mã hóa.

```
predictor = [0 1]; %  $y(k)=x(k-1)$ 
```

```
partition = [-1:1:.9];
```

```
codebook = [-1:1:1];
```

```
t = [0:pi/50.2*pi];
```

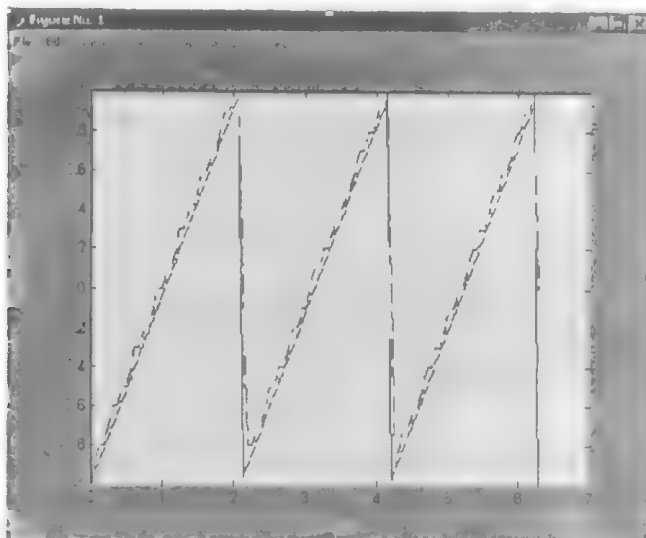
```
x = sawtooth(3*t); % Tín hiệu gốc
```

```
% Lượng tử hóa x sử dụng DPCM.
```

```

encodedx = dpcmenco(x,codebook,partition,predictor),
    % Cố gắng khôi phục x từ tín hiệu đã điều chế.
decodedx = dpcmdeco(encodedx,codebook,predictor),
plot(t,x,t,decodedx,'--')
distor = sum((x-decodedx).^2)/length(x) % Sai số trung bình bình phương
distor =
    0.0327

```



Hình 23.6

Một ví dụ hoàn chỉnh

Các phần này trình bày và diễn giải các phần của mã ví dụ:

- "Thiết lập các tham số"
- "Tạo ra tín hiệu"
- "Điều chế tín hiệu"
- "Thêm vào nhiễu"
- "Điều chế tín hiệu"
- "Tính toán và hiển thị các tỉ số lỗi bit"
- "Vẽ một chòm sao tín hiệu".

Thiết lập các tham số

Phần đầu tiên của ví dụ xác định các biến mà phần còn lại của tín hiệu sử dụng. Bảng chữ cái symbol có M symbol khác nhau, có tên với số nguyên nằm giữa 0 và M-1. Thông điệp là một vector cột có các đầu vào len, mỗi cái trong chúng chọn một trong các chữ cái symbol.

Các biến Fd và Fs trở đến tốc độ lấy mẫu tương đối của sơ đồ điều chế. Nó sẽ có đầy đủ ý nghĩa hơn nếu ví dụ là việc lấy mẫu một tín hiệu thực mà có giá trị tự nhiên theo thời gian. Tuy nhiên, vì ví dụ này sử dụng một tín hiệu ngẫu nhiên vốn không có đặc tính bền trong theo thời gian, mục đích chính của Fd và Fs là chỉ thị các tín hiệu đã điều chế có ba đầu vào cho mỗi đầu vào của tín hiệu gốc.

% Thiết lập các tham số.

M = 8; % Số các symbol trong bảng chữ cái

len = 10000; % Số các symbols trong thông điệp gốc

Fd = 1; % Giả thiết thông điệp gốc được lấy mẫu

% ở tốc độ 1 mẫu trong một giây.

Fs = 3; % Tín hiệu điều chế sẽ được lấy mẫu ở

% tốc độ 3 mẫu trong một giây.

Tạo tín hiệu

Biến signal là một ma trận $len \times 1$ chính là một vector cột chiều dài len, với các đầu vào là các số nguyên chọn ngẫu nhiên giữa 0 và M-1. Đây là tín hiệu cần điều chế trong ví dụ. Hàm rand nt là một phần của công cụ này.

% Tạo ra một tín hiệu

signal = randint(len,1,M); % thông điệp số hóa ngẫu nhiên

% chứa các số nguyên nằm giữa 0 and M-1

Điều chế tín hiệu

Phần này của ví dụ điều chế dữ liệu trong vector cột signal theo hai cách. Hàm dmodce thực hiện cả hai sự điều chế và đặt các kết quả trong ma trận hai cột modsignal.

Lần gọi dmodce đầu tiên, vốn tạo ra cột đầu tiên của modsignal, yêu cầu dmodce sử dụng điều chế QASK trên M symbol. Chuỗi 'qask' chỉ ra phương pháp QASK là cấu hình chòm sao hình vuông mặc định. Ở trường hợp này, nó cài đặt mã Gray.

Lần gọi dmodce thứ hai, vốn tạo ra cột thứ hai của modsignal, yêu cầu dmodce dùng điều chế QASK với một chòm sao tín hiệu mà các cấu hình của nó được đại diện trong hai vector inphase và quadr. Các biến inphase và quadr là các vector có chiều dài M

mà theo thứ tự liệt kê các thành phần trong pha và cầu phương của các điểm trong chòm sao tín hiệu. Các điểm được liệt kê theo chuỗi, để kết hợp một symbol thông điệp ở k với phần tử thứ $(k+1)$ trong inphase và quadr. Trong khi mã Gray đánh nhãn các điểm chòm sao theo một cách đặc biệt, cấu hình này liệt kê các điểm theo một chuỗi mà đơn thuần thuận tiện trong việc tạo inphase và quadr

Các dòng này cũng thể hiện vài cách chung để tạo ra các ma trận trong MATLAB. Nếu bạn không quen thuộc với dấu hai chấm trong MATLAB hoặc các hàm như ones và zeros; thì bạn cần xem xét lại các tài liệu MATLAB.

```
% Sử dụng điều chế M-ary QASK với hai chòm sao
% được đánh nhãn khác nhau.
modsignal(:,1) = dmodce(signal,Fd,Fs,'qask',M);
inphase = [-3:2:3 -3:2:3];
quadr = [ones(1,4) -1*ones(1,4)];
modsignal(:,2) = dmodce(signal,Fd,Fs,'qask/arb',inphase,quadr);
```

Thêm nhiễu

Theo như định nghĩa điều chế QASK baseband, modsignal là một ma trận phức có $\text{len} \cdot F_s / F_d$ hàng và hai cột. Lệnh dưới đây thêm các số ngẫu nhiên phân bố thường vào các thành phần thực và ảo của modsignal, để tạo ra tín hiệu có nhiễu noisy. Hàm randn là một hàm nội tại của MATLAB.

Chú ý rằng câu lệnh cộng thêm vào modsignal một ma trận hoàn toàn thực kích thước thích hợp và một ma trận hoàn toàn ảo kích thước thích hợp. Sử dụng một vòng lặp để thêm nhiễu vào các đầu vào vô hướng riêng biệt của modsignal sẽ ít có hiệu quả hơn, bởi vì MATLAB được tối ưu cho các phép toán trên ma trận.

```
% Thêm nhiễu vào phần thực và ảo của tín hiệu điều chế
noisy = modsignal+.5*randn(len*Fs/Fd,2)..
+j*.5*randn(len*Fs/Fd,2),
```

Giải điều chế tín hiệu

Phần này của ví dụ thực hiện giải điều chế tín hiệu điều chế có nhiễu noisy theo hai cách. Hàm ddemodce thực hiện đồng thời sự giải điều chế bởi thực hiện riêng rẽ trên mỗi cột của noisy. Trong mỗi trường hợp, ddemodce đặt các kết quả vào ma trận hai cột newsignal

```
% Giải điều chế để khôi phục bản tin.
newsignal(:,1) = ddemodce(noisy(:,1),Fd,Fs,'qask',M);
```

```
newsignal(:,2) = ddemodce(noisy(:,2),Fd,Fs,...
'qask/arb',inphase,quadr);
```

Tính toán và hiển thị tỉ số lỗi bit

Hàm `biterr` so sánh mỗi tín hiệu đã giải điều chế (tức là mỗi cột của `newsignal`) với tín hiệu gốc. Sau đó `biterr` tính số các lỗi bit, cũng như là tỉ số của lỗi bit. Hàm nội tại MATLAB `disp` hiển thị hai giá trị tỉ số lỗi bit trong cửa sổ dòng lệnh MATLAB.

```
% Kiểm tra xem liệu mã hóa Gray cho ít lỗi bit hơn.
% So sánh tín hiệu với mỗi cột của tín hiệu mới.
[num,rate] = biterr(newsignal,signal);
disp('Bit error rates for the two constellations used here')
disp('-----')
disp(['Gray code constellation:      ', num2str(rate(1))])
disp(['Non-Gray code constellation: ', num2str(rate(2))])
```

Vẽ chòm sao tín hiệu

Hàm `modmap` vẽ và gán nhãn chòm sao tín hiệu vuông mặc định có `M` điểm. Chòm sao do `inphase` và `quadr` xác định trông giống nhau, ngoại trừ các điểm được gán nhãn từ trái sang phải ngang qua mỗi hàng trong lược đồ, bắt đầu từ hàng trên.

```
% Vẽ các chòm sao tín hiệu với việc gán nhãn mã hóa Gray.
modmap('qask',M);
```

Các kết quả ra trong ví dụ

Ví dụ tạo ra các kết quả ra trong cửa sổ dòng lệnh MATLAB như chỉ ra dưới đây. Bởi vì tín hiệu bản tin và nhiễu là ngẫu nhiên, bạn sẽ có khả năng không nhận được chính xác các con số như bên dưới (xem thêm về các chuỗi trạng thái và lặp lại của các số ngẫu nhiên, xem trang đối chiếu về hàm nội tại MATLAB `rand`).

```
Bit error rates for the two constellations used here
(tỉ lệ lỗi bit cho hai chòm sao dùng ở đây)
```

```
-----
Gray code constellation: 0.0003
```

```
Non-Gray code constellation: 0.00036667
```

Ví dụ cũng tạo ra một cửa sổ figure chứa chòm sao tín hiệu vẽ trong hình bên dưới. Trục ngang đại diện cho các thành phần in-phase và trục dọc đại diện cho các thành phần cầu phương. Các chấm là các điểm của chòm sao. Số cạnh mỗi điểm là các symbol bản tin kết hợp với điểm đó. Nhờ xem xét dạng nhị phân của mỗi số từ 0 đến M-1, bạn có thể kiểm tra thấy rằng chòm sao này cái đặt mã Gray.

23.23 Phân loại các hàm

Communications Toolbox phân loại các hàm theo những mục sau:

- Các nguồn tín hiệu
- Các hàm phân tích tín hiệu
- Mã hóa nguồn
- Mã hóa kiểm soát lỗi
- Các hàm mức thấp cho mã hóa điều khiển lỗi
- Điều chế và giải điều chế
- Các bộ lọc đặc biệt
- Các hàm mức thấp cho các bộ lọc đặc biệt
- Các hàm kênh
- Các hàm tính trường Galois
- Các hàm tính trong các trường Galois đặc tính riêng
- Các tiện ích

Các nguồn tín hiệu

randerr	Sinh ra các mẫu lỗi bit
randint	Sinh ra ma trận các số nguyên phân bố ngẫu nhiên đều
randsrc	Sinh ra ma trận ngẫu nhiên sử dụng bảng chữ cái quy định
wgn	Sinh ra nhiễu trắng Gaussian

Các hàm phân tích tín hiệu

biterr	Tính số lỗi bit và tốc độ lỗi bit
eyediagram	Sinh ra 1 đồ thị mắt

scatterplot	Sinh ra 1 đồ thị phân tán (đồ thị rải)
symerr	Tính số lỗi kí hiệu và tốc độ lỗi kí hiệu

Mã hóa nguồn

arithdeco	Giải mã mã nhị phân sử dụng giải mã số học
arithenco	Mã hóa 1 chuỗi kí hiệu sử dụng mã hóa số học
compand	Bộ nén hay dẫn mã nguồn luật A hay luật μ
dpcmdeco	Giải mã sử dụng điều chế mã xung vi sai
dpcmenco	Mã hóa sử dụng điều chế mã xung vi sai
dpcmopt	Tối ưu các tham số điều chế mã xung vi sai
lloyds	Tối ưu hóa các tham số lượng tử sử dụng thuật toán Lloyd
quantiz	Sinh ra chỉ số lượng tử và giá trị đầu ra được lượng tử hóa

Mã hóa kiểm soát lỗi

bchpoly	Sinh ra các tham số hay các đa thức sinh cho mã nhị phân BCH
convenc	Dữ liệu nhị phân mã hóa chập
cyclgen	Sinh ra ma trận kiểm tra chẵn-lẻ và ma trận sinh cho mã vòng
cyclpoly	Sinh ra các đa thức sinh cho 1 mã vòng
decode	Bộ giải mã mã khối
encode	Bộ mã hóa mã khối
gen2par	Chuyển đổi giữa các ma trận kiểm tra chẵn lẻ và ma trận sinh
gfweight	Tính khoảng cách tối thiểu của 1 mã khối tuyến tính
hammgen	Sinh ra ma trận sinh và ma trận kiểm tra chẵn lẻ cho mã Hamming
rsdec	Bộ giải mã Reed-Solomon
rsdecof	Giải mã 1 file ASCII mã được mã hóa sử dụng mã Reed-Solomon
rsenc	Bộ mã hóa Reed-Solomon
rsencof	Mã hóa 1 file ASCII file sử dụng mã Reed-Solomon
rsgenpoly	Đa thức sinh của mã Reed-Solomon
syndtable	Sinh ra bảng giải mã đặc trưng
vitdec	Dữ liệu nhị phân giải mã chập sử dụng thuật toán Viterbi

Các hàm mức thấp cho mã hóa kiểm soát lỗi

bchdeco	Bộ giải mã BCH
bchenco	Bộ mã hóa BCH

Điều chế và giải điều chế

ademod	Bộ giải điều chế băng thông tương tự
ademodce	Bộ giải điều chế băng gốc tương tự
amod	Bộ điều chế băng thông tương tự
amodce	Bộ điều chế băng gốc tương tự
apkconst	Vẽ đồ thị chòm sao của tín hiệu ASK-PSK vòng kết hợp
ddemod	Bộ giải điều chế băng thông số
ddemodce	Bộ giải điều chế băng gốc số
demodmap	Tái tạo 1 bản tin số từ 1 tín hiệu điều chế
dmod	Bộ điều chế băng thông số
dmodce	Bộ điều chế băng gốc số
modmap	Chuyển 1 tín hiệu số sang 1 tín hiệu tương tự
qaskdeco	Tạo lại 1 bản tin từ chòm sao tín hiệu điều chế vuông QASK
qaskenco	Chuyển đổi 1 bản tin sang 1 đồ thị chòm sao tín hiệu điều chế vuông QASK

Các bộ lọc đặc biệt

hank2sys	Chuyển đổi 1 ma trận Hankel sang 1 mô hình hệ thống tuyến tính
hilbiir	Thiết kế 1 bộ lọc IIR hàm truyền Hilbert
rcosflt	Lọc tín hiệu vào sử dụng bộ lọc cos nâng
rcosine	Thiết kế 1 bộ lọc cos nâng

Các hàm mức thấp cho các bộ lọc đặc biệt

rcosfir	Thiết kế 1 bộ lọc FIR cos nâng
rcosiir	Thiết kế 1 bộ lọc IIR cos nâng

Các hàm kênh

awgn	Cộng nhiễu trắng Gaussian vào tín hiệu
------	--

Các phép tính trường Galois

<code>+</code>	Phép cộng và trừ các dãy Galois
<code>*</code>	Phép nhân và chia ma trận của các dãy Galois
<code>./</code>	Phép nhân và chia phần tử của các dãy Galois
<code>^</code>	Phép lấy mũ ma trận của dãy Galois
<code>.^</code>	Phép lấy mũ phần tử của dãy Galois
<code>'</code>	Hoán vị dãy Galois
<code>==, ~=</code>	Các toán tử quan hệ cho dãy Galois
<code>all</code>	True nếu tất cả các phần tử của 1 vector Galois khác không
<code>any</code>	True nếu bất kỳ phần tử nào của vector Galois khác không
<code>conv</code>	Thực hiện chập các vector Galois
<code>convmtx</code>	Ma trận chập của vector trường Galois
<code>cosets</code>	Sinh ra các tập hợp cyclotomic cho một trường Galois
<code>deconv</code>	Giải chập và chia đa thức
<code>det</code>	Tính định thức của ma trận vuông Galois
<code>dftmtx</code>	Ma trận chuyển đổi Fourier rời rạc trong một trường Galois
<code>diag</code>	Các ma trận đường chéo Galois và các đường chéo của một ma trận Galois
<code>fft</code>	Chuyển đổi Fourier rời rạc
<code>filter</code>	Bộ lọc số một chiều
<code>gf</code>	Tạo ra một ma trận trường Galois
<code>gftable</code>	Tạo ra 1 file để tăng tốc các phép tính trường Galois
<code>ifft</code>	Chuyển đổi ngược Fourier rời rạc
<code>inv</code>	Ma trận nghịch đảo của ma trận Galois
<code>isempty</code>	True nếu các dãy Galois trống
<code>isprimitive</code>	True nếu là đa thức gốc của 1 trường Galois
<code>length</code>	Chiều dài của vector Galois
<code>log</code>	Hàm tính loga cho 1 trường Galois
<code>lu</code>	Tìm ma trận thừa số tam giác trên và dưới của dãy Galois
<code>minpol</code>	Tìm đa thức tối thiểu của 1 phần tử của trường Galois.

mldivide	Mã trận kết quả của phép chia ma trận Galois \
polyval	Tính đa thức trong trường Galois
primpoly	Tìm các nguyên hàm cho trường Galois
rank	Hạng của một ma trận Galois
reshape	Thay đổi kích thước của ma trận Galois
roots	Tìm nghiệm đa thức
size	Xác định kích thước của ma trận, mảng
tril	Lấy ra ma trận tam giác chéo dưới của 1 ma trận
triu	Lấy ra ma trận tam giác chéo trên của 1 ma trận
gfadd	Cộng đa thức trường Galois
gfconv	Nhân đa thức trường Galois
gfcosets	Tạo ra các tập cyclotomic cho 1 trường Galois
gfdeconv	Chia đa thức trường Galois
gfdiv	Chia các phần tử của trường Galois
gffilter	Lọc dữ liệu sử dụng các đa thức cho một trường nguyên tố Galois
gflineq	Tìm nghiệm của phương trình $Ax = b$
gfminpol	Tìm đa thức tối thiểu của 1 phần tử trường Galois
gfmul	Nhân các phần tử của 1 trường Galois
gfpretty	Hiển thị 1 đa thức trong dạng truyền thống (ví dụ: $1 + x^2$)
gfprimck	Kiểm tra 1 đa thức có phải là không rút gọn được không
gfprimdf	Tạo ra các đa thức không rút gọn được mặc định cho 1 trường Galois
gfprimfd	Tìm các đa thức không rút gọn cho một trường Galois
gfrank	Tính hạng của 1 ma trận trong 1 trường Galois
gfrepconv	Chuyển 1 dạng biểu diễn nhị phân sang dạng khác
gfroots	Tìm các thừa số của 1 đa thức trường Galois
gfsb	Trừ đa thức trường Galois
gftrunc	Tối thiểu chiều dài của 1 dạng biểu diễn đa thức

gftuple	Đơn giản hóa hoặc chuyển đổi định dạng các phần tử trường Galois
bi2de	Chuyển đổi các vector nhị phân thành các số thập phân
de2bi	Chuyển đổi các số thập phân thành các vector nhị phân
erf	Hàm lỗi
erfc	Hàm lỗi bù ($1 - \text{erf}$)
istrellis	Kiểm tra xem đầu vào có ở cấu trúc trellis hợp lệ không
marcumq	Hàm Marcum Q tổng quát
mask2shift	Chuyển đổi vector mặt nạ sang dạng dịch trong thiết lập thành ghi dịch
oct2dec	Chuyển các số cơ số 8 thành các số thập phân
poly2trellis	Chuyển các đa thức mã chập thành dạng mô tả trellis
shift2mask	Chuyển đổi dạng dịch sang vector mặt nạ trong thiết lập thành ghi dịch
vec2mat	Chuyển 1 vector thành 1 ma trận

TRỢ GIÚP

24.1 Cửa sổ lệnh trợ giúp

MATLAB trợ giúp một số lệnh giúp bạn truy nhập thông tin nhanh chóng về các lệnh của MATLAB hoặc các hàm bên trong cửa sổ lệnh, bao gồm *help*, *lookfor*, *whatsnew*, và *info*.

24.1.1 Lệnh *help*

Lệnh *help* của MATLAB là cách đơn giản nhất để nhận trợ giúp nếu bạn biết được topic của cái cần trợ giúp. Nhập vào lệnh *help* topic, màn hình sẽ hiển thị nội dung của topic đó nếu như nó tồn tại. Ví như:

```
>> help sqrt
```

```
SQRT Square root.
```

```
SQRT(x) is the square root of the elements of x. complex results are produced if  
X is not positive
```

```
See also SQRT
```

Bạn sẽ nhận được trợ giúp của MATLAB về hàm căn bậc hai. Mặt khác, nếu như bạn nhập vào dòng lệnh sau:

```
>> help cows
```

```
cows not found
```

thì MATLAB sẽ không biết gì về cows. Bởi vì hàm này không có trong thư viện mẫu.

Chú ý: trong ví dụ trên, SQRT được viết chữ hoa. Tuy nhiên khi sử dụng sqrt không bao giờ là chữ in, do MATLAB là một ngôn ngữ chặt chẽ nên SQRT sẽ không được biết đến và quá trình sẽ sinh ra lỗi.

```
>> SQRT (2)
```

??? SQRT (

|

Missing operator, coma, or semicolon.

Để tóm tắt, tên hàm được in hoa để cho dễ đọc nhưng khi sử dụng, hàm sử dụng kí tự chữ thường.

Lệnh **help** hoạt động tốt nếu nếu như bạn biết chính xác topic mà bạn muốn trợ giúp mà điều này thường khó thực hiện, **help** hướng dẫn bạn trực tiếp truy tìm chính xác các topic mà bạn muốn, bạn chỉ đơn giản nhập vào **help** mà không có topic.

>> help

HELP topics

MATLAB: general	- mục đích chung của câu lệnh
MATLAB: ops	- các toán tử và các kí hiệu đặc biệt
MATLAB: lang	- xây dựng ngôn ngữ lập trình.
MATLAB: elphun	- các hàm toán học sơ đẳng
MATLAB: specfun	- các hàm toán học đặc biệt
MATLAB: matfun	- hàm ma trận - đại số học tuyến tính
MATLAB: datafun	- hàm biến đổi fourier và phân tích dữ liệu
MATLAB: polyfun	- các đa thức và phép nội suy
MATLAB: funfun	- phương án giải các ODE và các hàm của hàm
MATLAB: sparfun	- ma trận sparfun
MATLAB: graph2d	- đồ hoạ 2 chiều
MATLAB: graph3d	- đồ hoạ 3 chiều
MATLAB: specgraph	- đồ thị phổ
MATLAB: graphics	- thao tác đồ hoạ
MATLAB: uitools	- các công cụ giao tiếp người sử dụng và đồ hoạ
MATLAB: strfun	- xử lí kí tự
MATLAB: iofun	- tệp vào / ra
MATLAB: timefun	- ngày tháng và thời gian

MATLAB: datatypes	- cấu trúc và kiểu dữ liệu
MATLAB: MacOS	- các hàm trong Macintosh
MATLAB: demos	- ví dụ và minh hoạ
MATLAB: specmat	- ma trận đặc biệt
MATLAB: local	- tham chiếu
MATLAB: control	- hộp công cụ hệ thống điều khiển
MATLAB: signal	- hộp công cụ xử lý tín hiệu
MATLAB: symbolic	- hộp công cụ toán học

Thêm trợ giúp trong thư mục: topic, nhập vào " **help topic**".

24.1.2 Lệnh lookfor

Lệnh **lookfor** cung cấp sự trợ giúp bằng việc tìm kiếm tất cả các dòng đầu của **help topic**, và các M-file trên đường dẫn mà MATLAB đang tìm, và trả lại danh sách tất cả các file chứa từ khoá mà bạn khai báo. Một điều rất quan trọng là từ khoá không cần thiết là một lệnh của MATLAB. Ví dụ:

```
>> lookfor complex

CONJ      complex conjugate

IMAG      complex imaginary part

REAL      complex real part

CDF2RDF    complex diagonal form to real block diagonal form

RSF2CSF    real block diagonal form to complex diagonal form

CP_XPAIR   sort numbers into complex conjugate pairs
```

Từ khoá **complex** không phải là một lệnh của MATLAB, nhưng nó vẫn được tìm ra ở phần **help** gồm 6 lệnh của MATLAB. Nếu muốn biết thông tin về các lệnh này, hãy nhập vào từ lệnh **help**. Ví dụ:

```
>> help CONJ

CONJ      complex

CONJ(x) is the complex conjugate of X
```

For a complex x , $\text{CONJ}(X) = \text{REAL}(X) - i * \text{IMAG}(X)$

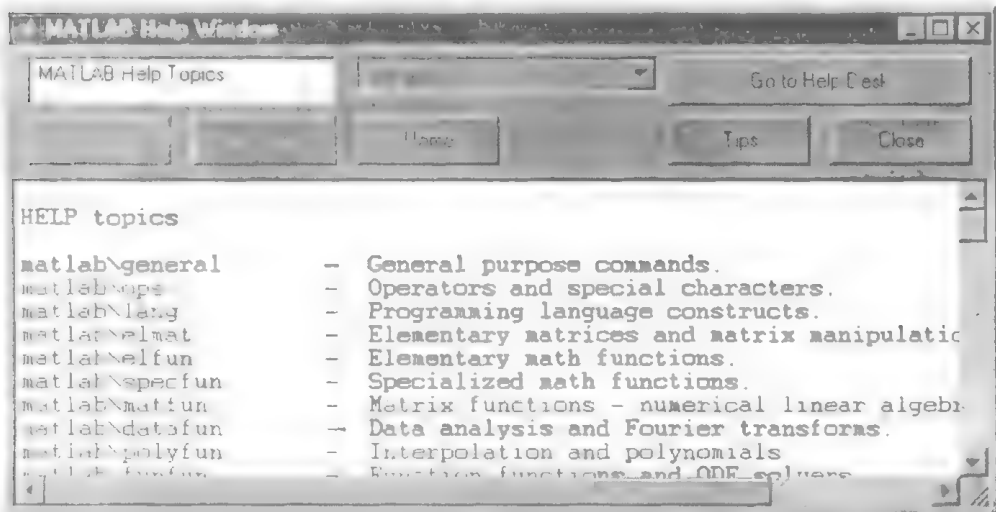
See also: **REAL**, **IMAG**, **I**, **J**

24.1.3 Lệnh *whatsnew* và *info*

Đúng như tên gọi của nó, **whatsnew** và **info** hiển thị những thông tin về những thay đổi và những sự cải tiến MATLAB và hộp dụng cụ của nó, nếu dùng mà không có đối số, thì **info** sẽ hiển thị những thông tin chung về MATLAB, phương pháp tiếp cận MathWorks, còn nếu dùng có đối số, ví như: **whatsnew** MATLAB hoặc **info signal**, thì file **Readme** chứa thông tin **Toolbox** sẽ hiển thị, nếu nó tồn tại.

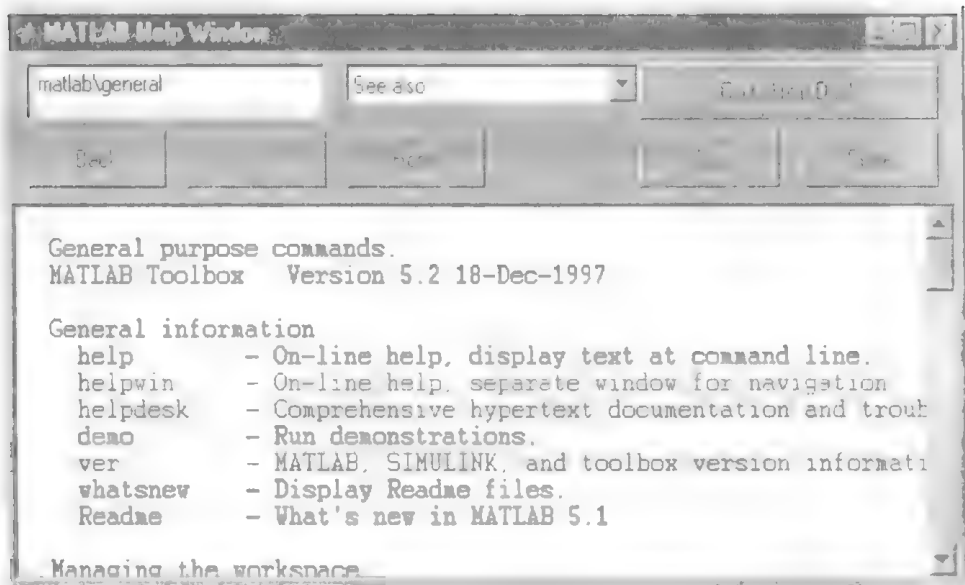
24.2 Cửa sổ trợ giúp

Một sự mở rộng của hệ thống trợ giúp trong MATLAB5 đó là cửa sổ **help** mới. Lệnh **helpwin** sẽ mở ra cửa sổ mới trên màn hình của bạn và bạn có thể dùng chuột để di chuyển thanh sang đến mục nào mà bạn quan tâm. Nếu dùng lệnh **helpwin** mà không có tham số, thì cửa sổ **help** có dạng như hình sau:



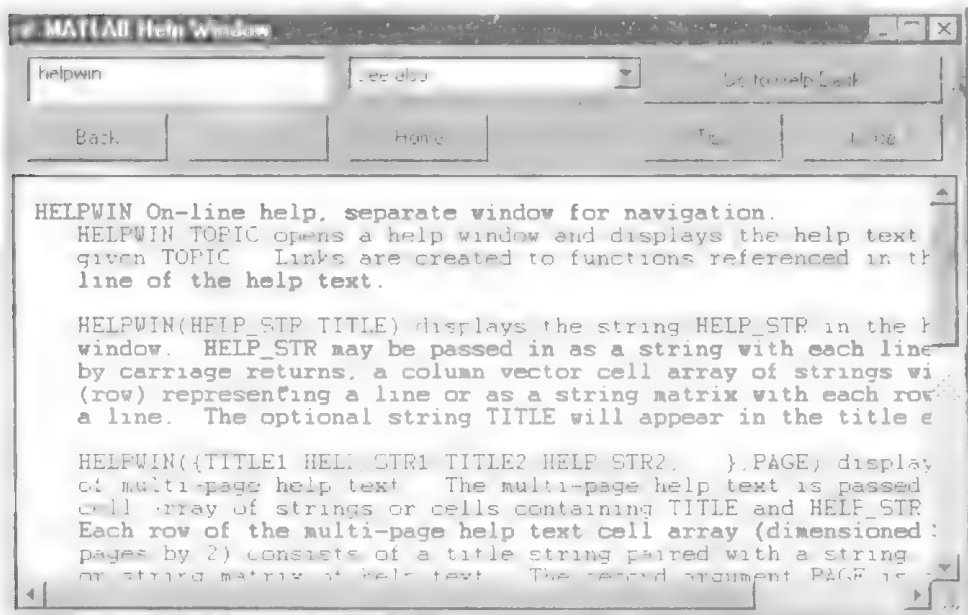
Hình 24.1 Cửa sổ trợ giúp

Nhấn vào vào bất cứ topic nào hiển thị trong cửa sổ **help**, sẽ hiển thị một cửa sổ mới chứa các topic con hoặc các hàm gắn với nó (hình 24.2).



Hình 24.2

Nhấn kép vào bất cứ biểu tượng nào trong đó sẽ hiển thị thông tin trợ giúp về mục đó (hình 24.3).



Hình 24.3

24.3. Các M- File của Student Edition

HELP Topic

Thư mục MATLAB	
Toolbox / local	Quyền ưu tiên
MATLAB / general	mục đích chung của câu lệnh
MATLAB / ops	Các toán tử và các kí tự đặc biệt
MATLAB / lang	Xây dựng ngôn ngữ lập trình
MATLAB / elmat	Thao tác ma trận và ma trận cơ sở
MATLAB / efun	Các hàm toán học cơ sở
MATLAB / specfun	Các hàm toán học đặc biệt
MATLAB / matfun	Đại số học tuyến tính - các hàm ma trận
MATLAB / datdfun	Bên đổi Fourier và phân tích dữ liệu
MATLAB / polyfun	Đa thức và các phân thức
MATLAB / lfun	Phương án giải các ODE và các hàm của hàm
MATLAB / sparsfun	ma trận sparsfun
MATLAB / graph2d	Đồ họa 2 chiều
MATLAB / graph3d	Đồ họa 3 chiều
MATLAB / specgraph	Đồ thị phổ
MATLAB / graphics	Thao tác đồ họa
MATLAB / ui / ios	Các công cụ giao tiếp người sử dụng và đồ họa
MATLAB / strfun	Xâu kí tự
MATLAB / iofun	Tệp vào / ra
MATLAB / timefun	Ngày tháng và thời gian
MATLAB / datatypes	Cấu trúc và kiểu dữ liệu
MATLAB / MacOS	Các hàm trong Macintosh
MATLAB / demos	Ví dụ và minh họa
MATLAB / specialmat	Ma trận đặc biệt

MATLAB / local	Tham chiếu
MATLAB / control	Hộp công cụ hệ thống điều khiển
MATLAB / signal	Hộp công cụ xử lý tín hiệu
MATLAB / symbolic	Hộp công cụ toán học

Tập tham chiếu	
startup	Tạo file khởi tạo
finish	Tạo file khoá
MATLABrc	File khởi tạo chủ
pathdef	Tìm đường dẫn mặc định
docopt	Web duyệt qua các mặc định
printopt	Máy in mặc định

Các lệnh tham chiếu	
cedit	Thiết lập dòng lệnh soạn thảo
terminal	Thiết lập đầu cuối đồ hoạ
colordef	Thiết lập màu mặc định
graymon	Thiết lập cửa sổ đồ hoạ mặc định cho loại màn hình cân chỉnh độ sáng
whitebg	Thay đổi màu nền của trục

Thông tin cấu hình	
hostid	Chỉ số nhận diện các MATLAB server chủ
license	Chỉ số đăng kí
version	Chỉ số phiên bản MATLAB

Mục đích chung của câu lệnh

Quản lý thông tin chung	
help	Trợ giúp trực tuyến, hiển thị văn bản tại các dòng lệnh
help win	Trợ giúp trực tuyến, cửa sổ truy xuất
help desk	Tra nhanh thông tin và các thắc mắc
demo	Chạy các chương trình mẫu
whatsnew	Hiển thị các file Readme
Readme	Thông tin mới cập nhật ở MATLAB 5

Quản lý không gian làm việc	
who	Danh sách các biến hiện tại
whos	Danh sách các biến hiện tại, khuôn dạng dài
clear	Xoá bỏ các biến và hàm khỏi bộ nhớ
pack	Hợp nhất không gian làm việc
load	Nạp các biến vào không gian làm việc từ đĩa
save	Lưu các biến vào đĩa
quit	Thoát khỏi mục hiện tại MATLAB

Quản lý đường dẫn	
path	Nhận/tạo đường dẫn
addpath	Thêm thư mục theo đường dẫn
rmpath	Rời thư mục từ đường dẫn
setpath	Sửa đổi đường dẫn

Câu lệnh điều khiển	
echo	Lấy lại lệnh từ file-M
more	Kiểm soát đầu ra các trang ở cửa sổ lệnh
diary	Lưu giữ văn bản
format	Thiết lập định dạng cho đầu ra

Hoạt động của lệnh hệ thống	
cd	Thay đổi thư mục làm việc hiện tại
pwd	Hiển thị thư mục làm việc hiện tại
dir	Danh sách thư mục
delete	Xoá file
getenv	Lấy lại biến môi trường
!	Thực hiện câu lệnh của hệ điều hành
dos	Thực hiện lệnh dos và trả lại kết quả
unix	Thực hiện lệnh unix và trả lại kết quả
vms	Thực hiện lệnh VMS DCL và trả lại kết quả
web	Mở trình xét duyệt Web
computer	Loại máy tính

M-file gỡ rối	
debug	Danh sách các lệnh gỡ rối
dbstop	Tạo điểm ngắt
dbclear	Di chuyển điểm ngắt
dbcont	Tiếp tục thực hiện lệnh
dbstack	Hiển thị các hàm gọi ngăn xếp
dbstatus	Danh sách các điểm ngắt
dbstep	Thực hiện một hoặc nhiều dòng
dbtype	Danh sách file-M với số lượng dòng
dbup	Thay đổi phạm vi không gian làm việc địa phương
dbquit	Thoát khỏi chế độ gỡ rối
dbmex	file- MEX gỡ rối (chỉ cho UNIX)

Các toán tử và các kí tự đặc biệt

Các toán tử	
plus (+)	Cộng
uplus (+)	Cộng unary
minus (-)	Trừ
uminus (-)	Trừ unary
mtimes(. *)	Nhân ma trận
times (*)	Nhân mảng
mpower (^)	Lũy thừa ma trận
power (. ^)	Lũy thừa mảng
mdivide (\)	Chia trái ma trận
mrdivide ./	Chia phải ma trận
ldivide (.\)	Chia trái mảng
ndivide (./)	Chia phải ma trận
kron	Sản phẩm cơ căng

Toán tử quan hệ	
eq (==)	Bằng
ne (~=)	Không bằng
lt (<)	Nhỏ thua
gt (>)	Lớn hơn
le (<=)	Nhỏ thua hoặc bằng
ge (>=)	Lớn hơn hoặc bằng

Toán tử logic	
and (&)	Logic và

or ()	Logic hoặc
not (~)	Logic phủ định
xor	Logic hoặc phủ định
any	True nếu mọi phần tử của vector khác không
all	True nếu tất cả các phần tử khác không

Các toán tử Bitwise	
bitand	Bitwise and
bitcmp	Bit hoàn thành
bitor	Bitwise OR
bitmax	Maximum floating point integer
bitset	Thiết lập bit
bitget	Nhận bit
bitshift	Dịch bit

Thiết lập các kí tự	
union	Thiết lập liên kết
unique	Chỉ sử duy nhất
intersect	Thiết lập sự giao nhau
setdiff	Tạo sự khác nhau
setxor	Thiết lập hoặc phủ định
ismember	True nếu thiết lập các thành viên

Các kí tự đặc biệt	
colon	Dấu hai chấm
()	Dấu ngoặc đơn

[]	Dấu ngoặc vuông
{ }	Dấu ngoặc nhọn
.	Chấm thập phân
.	Truy nhập cấu trúc trường
..	Thư mục mẹ
...	Sự tiếp tục
,	Dấu phẩy
;	Dấu chấm phẩy
%	Chú thích
!	Liên quan câu lệnh của hệ điều hành
=	Gán
'	Nhảy
transpose('.')	Chuyển vi
ctranspose('')	Chuyển vi số phức liên hợp
horzcat [,]	Ghép chuỗi theo chiều ngang
vertcat[;]	Ghép chuỗi theo chiều đứng
subsasgn	Gán subscripted
bsref	Tham chiếu subscripted
subsindex	Chỉ số subscripted

Cấu trúc ngôn ngữ lập trình

Câu lệnh điều khiển	
if	Điều kiện thực hiện câu lệnh
elseif	Dùng với if
else	Dùng với if
end	Kết thúc lệnh if, for, while

for	Lặp lại câu lệnh một số lần
while	vòng lặp while
break	Thoát khỏi vòng lặp for và while
return	Trở về từ hàm gọi
pause	Tạm dừng cho tới khi nhấn một phím bất kì
Thi hành và định giá	
eval	Thực hiện xâu với biểu thức MATLAB
feval	Thực hiện hàm chỉ ra bởi xâu
evalin	Định giá các biểu thức trong không gian làm việc
builtin	Thực hiện các hàm được tạo bởi phương pháp xếp chồng
assignin	Gán các biến trong không gian làm việc
run	Chạy script

Script, hàm, và các biến	
script	Về script MATLAB và file-M
function	Thêm hàm mới
global	Định nghĩa biến toàn cục
mfilename	Tên và các M-file đang thực hiện hiện tại
lists	Dấu phẩy phân chia các danh sách
exist	Kiểm tra xem các biến hoặc các hàm có được định nghĩa hay không
isglobal	True nếu là biến toàn cục

Thao tác với các đối số	
nargchk	Công nhận số lượng các đối số đầu vào
nargin	Số lượng hàm các đối số đầu vào

nargout	Số lượng hàm các đối số đầu ra
varargin	Danh sách các đối số đầu vào, độ dài các biến
varargout	Danh sách các đối số đầu ra, độ dài các biến
inputname	Tên đối số đầu vào

Hiển thị thông báo	
error	Hiển thị thông báo lỗi và hàm huỷ
warning	Hiển thị thông báo cảnh báo
lasterr	Thông báo lỗi trước
errortrap	Bỏ qua lỗi trong quá trình kiểm tra
disp	Hiển thị một mảng
fprintf	Hiển thị thông báo định dạng
sprintf	Ghi dữ liệu định dạng vào một chuỗi

Đầu vào tương hỗ	
input	Nhắc người sử dụng nhập vào
keyboard	Gọi bàn phím từ M-file
pause	Đợi người sử dụng nhập dữ liệu vào
uimenu	Tạo giao diện bảng chọn-người sử dụng
uicontrol	Tạo giao diện người điều khiển

Ma trận cơ bản và thao tác với ma trận

Ma trận cơ bản	
zeros	Mảng số không
ones	Mảng số 1
eye	Nhận dạng ma trận

repmat	Tái tạo và mảng
rand	Số ngẫu nhiên xấp xỉ đồng đều
randn	Số ngẫu nhiên xấp xỉ thông thường
linspace	Vector không gian tuyến tính
logspace	Vector không gian logarithm
meshgrid	Mảng X-Y cho đồ thị 3 chiều
.	Vector không gian thông thường và chỉ số trong ma trận

Thông tin mảng cơ sở	
size	Kích cỡ ma trận
length	Độ dài vector
ndims	Số chiều
disp	Hiển thị ma trận hoặc văn bản
isempty	True nếu là ma trận trống
isequal	True nếu ma trận là đồng nhất
isnumeric	True cho mảng số
islogical	True cho mảng logic
logical	Chuyển đổi giá trị số thành logic

Thao tác với ma trận	
reshape	Thay đổi kích cỡ
diag	Ma trận đường chéo và đường chéo của ma trận
tril	Trích phía dưới ra ma trận tam giác
triu	Trích phía trên ra ma trận tam giác
flipr	Ma trận flip theo hướng trái/phải
flipud	Ma trận flip theo hướng trên/dưới

flipdim	Ma trận flip dọc theo chiều khai báo
rot90	Quay đi một góc 90 độ
find	Tìm chỉ số phân tử khác không
end	Chỉ số cuối
sub2ind	Chỉ số tuyến tính từ multiple subscripts
ind2sub	Multiple subscripts từ chỉ số tuyến tính

Hằng và các biến đặc biệt	
ans	Trả lại kết quả khi biểu thức không được gán
eps	Viết dưới dạng dấu phẩy động
realmax	Số dấu phẩy động dương lớn nhất
realmin	Số dấu phẩy động dương nhỏ nhất
pi	3.1415926535897...
i, j	Đơn vị ảo
inf	Vô cùng
NaN	Không phải là một số
isNaN	True nếu NaN
isinf	True nếu số phân tử là không vô cùng
isfinite	True nếu số phân tử là vô cùng
flops	Đếm số chữ số sau dấu phẩy động

Các biến đặc biệt	
ans	Trả lại kết quả khi biểu thức không được gán
eps	Độ chính xác sau dấu phẩy động
pi	π
i, j	$\sqrt{-1}$

inf	∞
NaN	Không phải dạng số
clock	Đồng hồ tường
date	Ngày
flops	Đếm sự hoạt động của dấu phẩy động
nargin	Số lượng các đối số của hàm vào
nargout	Số lượng các đối số hàm ra

Các loại ma trận đặc biệt	
comban	Bầu bạn
diag	Đường chéo
eye	Nhân dạng
gallery	Bí mật
hadamar	Hadamard
hankel	Hankel
hilb	Hilbert
invhilb	Hilbert đảo
linspace	Vector
logspace	Vector
magic	Vuông Magic
meshdom	Thực hiện cho mesh plots
ones	Hằng
rand	Các phần tử ngẫu nhiên
toeplitz	Toeplitz
vander	Vandermonde
zeros	Không

Các hàm toán học thông thường

Các hàm lượng giác	
sin	Hàm sine
cos	Hàm cosine
tan	Hàm tangent
asin	Hàm arcsine
acos	Hàm arccosine
atan	Hàm arctangent
atan2	Hàm arctan góc phần tư
sinh	Sine hyperpolic
cosh	Cosine hyperpolic
tanh	Tangent hyperpolic
asinh	Arcsine hyperpolic
acosh	Arccosine hyperpolic
atanh	Arctangent hyperpolic

Các hàm toán học	
abs	Trị tuyệt đối hoặc biên độ số phức
angle	Góc pha
sqrt	Căn bậc hai
real	Phần thực
imag	Phần ảo
conj	Phức liên hợp
round	Làm tròn đến số nguyên gần nhất
fix	Làm tròn đến không
floor	Làm tròn đến âm vô cùng

ceil	Làm tròn lên vô cùng
sign	Hàm dấu
rem	Sử dụng lại hoặc các khối (modulus)
exp	Hàm mũ cơ sở e
log	Logarithm tự nhiên
log10	Log 10 cơ sở

Các hàm đặc biệt	
airy	Hàm airy
besseli	Hàm Bessel loại thứ nhất
besselj	Hàm Bessel loại thứ hai
besselk	Hàm Bessel loại thứ ba (hàm Hankel)
bessely	Sửa đổi hàm Bessel loại thứ nhất
beta	Sửa đổi hàm Bessel loại thứ hai
betainc	Hàm beta
betaln	Hàm beta không hoàn toàn
erf	Hàm logarithm beta
erfc	Hàm lỗi
ellipk	Hàm lỗi thành phần
ellipj	Phép tích phân elliptic
gamma	Hàm elliptic Jacobian
gammaaln	Hàm gamma
inverf	Hàm logarithm gamma
rat	Hàm lỗi ngược
	Xấp xỉ

Hàm lý thuyết số học	
factor	Hệ số sơ khai
isprime	True nếu là số sơ khai
primes	Danh sách các số sơ khai
gcd	Bộ chia chung lớn nhất
lcm	Phép nhân chung nhỏ nhất
rat	Xấp xỉ hữu tỉ
rats	Đầu ra hữu tỉ
perms	Sự hoán vị
nchoosek	Sự tổ hợp chập K của N phần tử

Đồ hoạ

Trang đồ hoạ	
plot	Đồ thị tuyến tính X-Y
loglog	Đồ thị loglog X-Y
semilogx	Đồ thị semi-log X-Y
semilogy	Đồ thị semi-log X-Y
polar	Đồ thị toạ độ cực
mesh	Mặt lưới không gian 3 chiều
contour	Đồ thị đường bao
meshdom	Miền trong của đồ thị lưới
bar	Biểu đồ hình chữ nhật
errorbar	Thêm vào errorbars
title	Tiêu đề đồ thị
xlabel	Nhãn trục x
ylabel	Nhãn trục y

grid	Kẻ đường lưới trong đồ thị
text	Văn bản ở vị trí bất kì
gtext	Văn bản ở vị trí con trỏ
input	Nhập đồ họa

Điều khiển của sổ đồ họa	
axis	Cân chỉnh trục tọa độ và hình dạng của nó
zoom	Có vào hoặc dân ra đồ thị
hold	Giữ đồ thị trên màn hình
shg	Hiện thị đồ thị lên màn hình
clg	Xoá đồ thị trên màn hình
subplot	Tách cửa sổ đồ họa

Đồ họa trong không gian 3 chiều

Lệnh đồ họa thông thường	
plot3	Vẽ đường thẳng và điểm trong không gian 3 chiều
mesh	Bề mặt không gian 3 chiều
surf	Tô màu bề mặt không gian 3 chiều
fill3	Điền đầy đa giác 3 chiều

Cân chỉnh màu	
colormap	Tra cứu bảng màu
caxis	Sự phân chia bảng màu già
shading	Chế độ làm bóng
hidden	Chế độ dấu các nét
brighten	Bảng tra cứu màu tối hoặc sáng

Chiếu sáng	
surf1	Làm bóng bề mặt không gian 3 chiều bằng chiếu sáng
lighting	Chế độ chiếu sáng
material	Chế độ phản chiếu tự nhiên
specular	Sự phản chiếu
diffuse	Sự phản chiếu khuếch tán
surfnorm	Bề mặt thông thường

Bảng màu	
hsv	Bảng giá trị màu bão hoà
hot	Bảng màu đen- trắng- đỏ - vàng
gray	Bảng màu chia theo độ xám tuyến tính
pink	Màu hồng
white	Màu trắng
bone	Màu xám pha lẫn xanh da trời

Điều chỉnh trực	
ax.s	Điều chỉnh hình dạng và độ phân chia
zoom	Có vào hoặc dân ra đồ thị
gr d	Đường kẻ lưới
box	Hộp chứa trực toạ độ
hold	Lưu đồ thị hiện tại
axes	Xây dựng trực ở một vị trí bất kỳ

Chú thích đồ họa	
title	Tiêu đề đồ họa
xlabel	Nhãn trục x
ylabel	Nhãn trục y
zlabel	Nhãn trục z
colorbar	Hiển thị thanh màu
text	Chú thích văn bản
gtext	Di văn bản đến vị trí chuột

Chuỗi kí tự

Khái quát chung	
char	Tạo một chuỗi kí tự
double	Chuyển chuỗi sang mã số kí tự
cellstr	Tạo mảng chuỗi tế bào từ mảng kí tự
blanks	Xâu rỗng
deblank	Di chuyển các râu rỗng
eval	Thực hiện râu với biểu thức MATLAB

Kiểm tra chuỗi	
schar	True nếu là chuỗi kí tự (xâu)
iscellstr	True nếu là mảng chuỗi tế bào
isletter	True nếu là chữ hoa trong bảng chữ cái
isspace	True nếu là kí tự rỗng

Các phép toán với chuỗi	
strcat	Kết nối xâu
strvcat	Kết nối đọc xâu
strcmp	So sánh chuỗi
strncmp	So sánh N kí tự đầu tiên của chuỗi
findstr	Tìm một xâu bên trong xâu khác
strjust	Mảng kí tự đồng đều
strrep	Thay thế chuỗi bằng chuỗi khác
strtok	Tìm thẻ bài trong chuỗi
upper	Chuyển chuỗi sang chữ hoa
lower	Chuyển chuỗi sang kí tự thông thường

Chuỗi và văn bản	
abs	Chuyển đổi từ chuỗi sang giá trị ASCII
num2str	Đổi từ số thành chuỗi
int2str	Đổi số nguyên sang chuỗi
settr	Thiết lập cờ để chỉ rằng ma trận đó là một chuỗi
sprintf	Đổi số sang chuỗi
hex2num	Chuyển đổi chuỗi từ hệ 16 sang dạng số

File input/output

Mở và đóng file	
fopen	Mở file
fclose	Đóng file

File nhị phân i/o	
fread	Đọc dữ liệu nhị phân từ file
fwrite	Viết dữ liệu nhị phân lên file

File i/o định dạng	
fscanf	Đọc dữ liệu đã định dạng từ file
fprintf	Ghi dữ liệu đã định dạng lên file
fgetl	Đọc dòng lệnh từ file, thay bằng dòng mới
fgets	Đọc dòng lệnh từ file, giữ nguyên dòng mới
input	Hiển thị để người dùng nhập vào

Vị trí file	
ferror	Kiểm tra trạng thái file
feof	Kiểm tra xem đã kết thúc file hay chưa
fseek	Thiết lập bộ chỉ thị vị trí file
ftell	Nhận từ bộ chỉ thị vị trí file
frewind	Rewind file

Các hàm xuất nhập file	
load	Nạp không gian làm việc từ file-MAT
save	Lưu giữ không gian làm việc vào file - MAT
dlmread	Đọc file phân định ASCII
dlmwrite	Ghi file phân định ASCII

Xuất nhập file ảo	
imread	Đọc phần ảo từ file đồ họa
imwrite	Ghi phần ảo lên file đồ họa
iminfo	Trả lại thông tin về file đồ họa

Xuất nhập file audio	
auwrite	Ghi file âm thanh NEXT/ SUN (". au")
auread	Ghi file âm thanh NEXT/ SUN (". au")
wavwrite	Ghi file Microsoft WAVE (". wav")
wavread	Đọc file Microsoft WAVE (". wav")

Cửa sổ lệnh I / O	
clc	Xoá cửa sổ lệnh
home	Đưa con trỏ về đầu văn bản
disp	Hiện thị màn hình
input	Thông báo cho người sử dụng nhập vào
pause	Đợi từ người sử dụng trả lời

Thời gian và ngày

Giờ và ngày hiện tại	
now	Giờ và ngày hiện tại hiển thị dạng số
date	Giờ và ngày hiện tại hiển thị dạng chuỗi
clock	Giờ và ngày hiện tại hiển thị dạng vector

Các hàm cơ bản	
<code>datenum</code>	Số ngày nối tiếp
<code>datestr</code>	Chuỗi thay thế ngày
<code>datevec</code>	Thành phần ngày tháng

Hàm ngày tháng	
<code>calendar</code>	Lịch
<code>weekday</code>	Ngày trong tuần
<code>eomday</code>	Kết thúc tháng
<code>datetick</code>	Dấu tick định dạng cho ngày tháng

Hàm đếm	
<code>cputime</code>	Thời gian cpu tính theo đơn vị giây
<code>tic, toc</code>	Bộ đếm ngừng hoạt động
<code>etime</code>	Thời gian thiết lập
<code>pause</code>	Dừng trong một giây

Kiểu dữ liệu và cấu trúc

Kiểu dữ liệu	
<code>double</code>	Chuyển đổi thành double
<code>sparse</code>	Tạo một ma trận không liên tục
<code>char</code>	Xây dựng mảng kí tự
<code>cell</code>	Tạo mảng tế bào
<code>struct</code>	Xây dựng hoặc chuyển đổi thành mảng cấu trúc
<code>uint8</code>	Chuyển đổi thành số nguyên không dấu 8 bit
<code>inline</code>	Xây dựng đối tượng INLINE

Hàm của mảng nhiều chiều	
cat	Mảng kết nối
ndims	Số chiều
ndgrid	Tạo thành mảng cho các hàm N-D và phép nội suy
permute	Phép nội suy số chiều của mảng
ipermute	Nghịch đảo phép nội suy số chiều của mảng
shftim	Chuyển dịch số chiều

Hàm của mảng tế bào	
cell	Tạo mảng tế bào
celldisp	Hiển thị nội dung của mảng tế bào
cellplot	Hiển thị thuật hoa mảng tế bào
num2cell	Chuyển đổi mảng số thành mảng tế bào
deal	Phân phát đầu vào đến đầu ra
cel2struct	Chuyển đổi mảng tế bào thành mảng cấu trúc
struct2cell	Chuyển đổi mảng cấu trúc thành mảng tế bào
isCell	True nếu là mảng tế bào

Hàm cấu trúc	
struct	Tạo hoặc chuyển đổi thành mảng cấu trúc
fieldsnames	Nhận tên trường cấu trúc
getfield	Nhận lại nội dung của trường cấu trúc
setfield	Thiết lập nội dung trường cấu trúc
isfield	True nếu trường ở trong mảng cấu trúc
istruct	True nếu là mảng tế bào

Trao đổi dữ liệu động

Hàm DDE	
ddeadv	Thiết lập bộ giám sát liên kết
ddeexec	Đưa xâu ra để thực hiện
ddeinit	Khởi tạo sự giao tiếp DDE
ddereq	Yêu cầu dữ liệu từ các ứng dụng
ddeterm	Kết thúc sự giao tiếp DDE
ddeunadv	Cởi bỏ bộ giám sát liên kết

Ví dụ và sự thể hiện

MATLAB/matrận	
intro	Giới thiệu phép toán ma trận cơ bản trong MATLAB
inverter	Giải thích ma trận đảo
matman.p	Giới thiệu phép nhân ma trận

Cửa sổ lệnh	
clc	Xoá cửa sổ lệnh
home	Đưa con trỏ về đầu dòng
format	Thiết lập dạng hiển thị kết quả
disp	Hiển thị ma trận hoặc văn bản
fprintf	In số được định dạng
echo	Cho phép gọi lại câu lệnh

General	
hlep	Phương tiện trợ giúp
demo	Chạy các chương trình mẫu
who	Danh sách các biến trong bộ nhớ
what	Danh sách các M-file trên đĩa
size	Số chiều của hàng và cột
lengh	Độ dài vector
clear	Xoá không gian làm việc
computer	Loại máy tính
^C	Hủy biến địa phương
quit	Kết thúc chương trình
exit	Tương tự như quit

(Lập trình) Programming và file-M	
input	Nhập số từ bàn phím
keyboard	Gọi bàn phím như M-file
error	Hiển thị thông báo lỗi
function	Định nghĩa hàm
eval	Văn bản được giải thích trong các biến
feval	Hàm định giá được gọi ra bởi chuỗi
echo	Cho phép gọi lại câu lệnh
exist	Kiểm tra xem có biến tồn tại hay không
casesen	Thiết lập độ nhạy của case
global	Định nghĩa các biến toàn cục
startup	Khởi tạo M-file
getenv	Nhận chuỗi môi trường
menu	Lựa chọn từ bảng chọn
etime	Elapsed time (không kể đến thời gian)

Các file m-file	
chdir	Đổi thư mục hiện tại
delete	Xoá file
diary	Ghi mục
dir	Thư mục của file trên đĩa
load	Nạp các biến từ file
save	Lưu các biến nên file
type	Liệt kê hàm hoặc file
what	Hiển thị các M-file trên đĩa
fprintf	Viết vào file
pack	Nén bộ nhớ qua save

Đa thức	
poly	Đa thức đặc trưng
roots	Nghiệm đa thức- phương pháp ma trận bầu bạn
roots1	Nghiệm đa thức- phương pháp Laguerre
polyval	Ước lượng đa thức
polyvalm	Ước lượng đa thức ma trận
conv	Phép nhân
deconv	Phép chia
residue	Khai triển đa thức
polyfit	Sự điều chỉnh độ chênh lệch đa thức

Các hàm ma trận và đại số tuyến tính

Phân tích ma trận	
norm	Chỉ tiêu ma trận hoặc vector
normest	Định giá ma trận hai chỉ tiêu
rank	Hạng ma trận
det	Định thức
trace	Tổng các phần tử trên đường chéo chính
null	Không gian trống
orth	Tính trực giao
rref	Rút gọn hàng theo hình bậc thang
subspace	Góc giữa hai số âm

Phép toán tuyến tính	
\ và /	Lời giải phép toán tuyến tính; sử dụng help slash
inv	Ma trận đảo
cond	Số điều kiện đối với ma trận đảo
condest	Định giá số điều kiện một chỉ tiêu
lu	Sử tìm thừa số LU
luinc	Tìm thừa số LU không hoàn thành

Giá trị duy nhất	
svd	Sự phân tích giá trị duy nhất
svds	Một số giá trị duy nhất
poly	Đa thức đặc trưng
polyeig	Vấn đề của đa thức
condeig	Số điều kiện với hy vọng
qz	Sử tìm thừa số cho hàm suy rộng
schur	Sự phân tích chuỗi

Hàm ma trận	
expm	Ma trận theo hàm mũ
logm	Ma trận logarithm
sqrm	Ma trận bậc hai
funm	Định giá chung hàm ma trận

Tìm thừa số tiêu chuẩn	
qrdelete	Xoá bỏ thư mục từ sự tìm thư mục QR
qrinsert	Gai vào thư mục trong sự tìm thừa số QR
rsf2csf	Mẫu đường chéo thực tới mẫu đường chéo phức tạp
balance	Cân bằng để tăng độ chính xác

Biến đổi fourier và phân tích dữ liệu

Phép toán cơ bản	
max	Thành phần lớn nhất
min	Thành phần nhỏ nhất
mean	Giá trị trung bình
median	Giá trị trung tuyến
std	Độ lệch góc chuẩn
sum	Tổng của các số hạng
prod	Kết quả của các phân tử
hist	Biểu đồ
trapz	Hình thang số nguyên
cumsum	Tổng tích lũy của các phần tử
cumprod	Kết quả tích lũy của các phân tử
cumtrapz	Số nguyên tích lũy bậc thang

Sai phân có hạn	
diff	Sai phân và đạo hàm xấp xỉ
gradient	Gradient xấp xỉ
del2	Laplacien rời rạc

Filtering and convolution (nếp, cuộn)	
filter	Bộ lọc số một chiều
filter2	Bộ lọc số 2 chiều
conv	Phép nhân đa thức và sự nén lại
conv2	Nén 2 chiều
convn	Nén n chiều
deconv	Giải nén và chia đa thức

Biến đổi Fourier	
fft	Biến đổi Fourier rời rạc
fft2	Biến đổi Fourier rời rạc 2 chiều
fftn	Biến đổi Fourier rời rạc n chiều
ifft	Biến đổi Fourier rời rạc ngược
ifft2	Biến đổi Fourier rời rạc hai chiều
ifftn	Biến đổi Fourier rời rạc n chiều

Đa thức và phép nội suy

Phép nội suy	
interp1	Phép nội suy một chiều (tra bảng)
interp1q	Phép nội suy tuyến tính một chiều nhanh

interpft	Phép nội suy một chiều sử dụng phương pháp FFT
interp2	Phép nội suy hai chiều (tra bảng)
interp3	Phép nội suy ba chiều (tra bảng)
interp _n	Phép nội suy n chiều (tra bảng)
griddata	Điều chỉnh bề mặt và lưới dữ liệu

Hàm và giải pháp ODE

Optimization and Root Finding	
fmin	Tối thiểu hàm một biến
fmins	Tối thiểu hàm vài biến
fzero	Tìm hàm một biến không

Numeric Integration	
quad	Tích phân định giá về số lượng, phương pháp trật tự thấp
quad8	Tích phân định giá về số lượng, phương pháp trật tự cao hơn
dblquad	Tích phân hai lần định giá về số lượng

Đối tượng hàm inline	
inline	Xây dựng đối tượng INLINE
argnames	Tên đối số
formula	Thế thức hàm
char	Chuyển đổi đối tượng INLINE thành mảng kí tự

Ma trận rời rạc

Các ma trận không liên tục cơ bản	
speye	Ma trận đồng nhất thức không liên tục
sprand	Ma trận ngẫu nhiên phân chia một cách không liên tục đồng nhất
sprandn	Ma trận ngẫu nhiên phân chia một cách không liên tục thông thường
sprandsy	Ma trận đối xứng ngẫu nhiên không liên tục
spdiags	Ma trận không liên tục được tạo thành từ đường chéo

Full to Sparse Conversion	
sparse	Tạo ma trận không liên tục
full	Chuyển đổi ma trận không liên tục thành ma trận đầy đủ
find	Tìm chỉ số các phần tử khác không
spconvert	Nhập vào từ định dạng ma trận không liên tục bên ngoài

TÀI LIỆU THAM KHẢO

- [1]. Introduction to MatLab 6 for Engineers 1ED, William J. Palm, December 2000.
- [2]. Applied Optimization with MatLab Programming, K. Venkataraman, November 2001.
- [3]. Guide to MatLab: For Beginners and Experienced Users, Brian R. Hunt Ronald, L. Lipsman, September 2001.
- [4]. Computational Statistics Handbook with MatLab, Wendy L. Martinez, Angel R. Martinez, September 2001.
- [5]. Computational Statistics Handbook with MatLab, Wendy L. Martinez, Angel R. Martinez, September 2002.
- [6]. An Engineer's Guide to MatLab, Edward B. Magrab, August 2002.
- [7]. An Introduction to Numerical Methods: A MatLab Approach, Abdelwahab Kharab, December 2001.
- [8]. MatLab Programming for Engineer (Second Edition), Stephen J. Chapman, November 2001.
- [9]. Student Edition of MatLab, Version 5.3 for Microsoft Window BK&CD Rom, MathWorks Staff, April 1999.
- [10]. Introduction to Numerical Method and Matlab: Implementations and Applications, Gerald W. Recktenwald, August 2000.
- [11]. Engineering Problem Solving with MatLab, Second Edition, Delores M. Etter D.M. Etter, August 1996.

LẬP TRÌNH MATLAB

Dành cho sinh viên khối khoa học và kỹ thuật

Tác giả: ThS. Nguyễn Hoàng Hải

ThS. Nguyễn Việt Anh

Chịu trách nhiệm xuất bản

PGS. TS Tô Đăng Hải

Biên tập

Ngọc Khuê

Vẽ bìa

Hương Lan

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

70 TRẦN HƯNG ĐẠO - HÀ NỘI

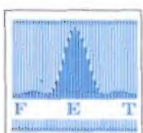
In 1.000 cuốn, khổ 16 x 24cm tại Xưởng in NXB Văn hoá Dân tộc
Quyết định xuất bản số: 409-2006/CXB/64.1-33/KHKT ngày 10/8/2006
In xong và nộp lưu chiểu Quý IV năm 2006.



1956 - 2006

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

**50 NĂM XÂY DỰNG
VÀ PHÁT TRIỂN**



1956 - 2006

KHOA ĐIỆN TỬ - VIỄN THÔNG

**50 NĂM HÌNH THÀNH
VÀ PHÁT TRIỂN**

T1 19 lập trình matlab và ứng



206317



Giá: 50.000đ